IBM Informix Guide to SQL

Reference

Informix Extended Parallel Server, Version 8.3
Informix Dynamic Server, Version 9.3

August 2001
Part No. 000-8338

Documentation Team: Jennifer Leland, Karin Moore, Judith Sherwood

Table of Contents

Introduction

In This Introduction 3
About This Manual 3
Types of Users 4
Software Dependencies 4
Assumptions About Your Locale. 5
Demonstration Database 5
New Features in Dynamic Server, Version 9.3 6
SQL Enhancements 6
Documentation Conventions 7
Typographical Conventions 7
Icon Conventions 8
Syntax Conventions 10
Sample-Code Conventions. 11
Additional Documentation 12
Related Reading 14
Compliance with Industry Standards 15
Informix Welcomes Your Comments 15

Chapter 1 System Catalog Tables

In This Chapter 1-5
Objects That the System Catalog Tables Track 1-5
Using the System Catalog 1-6
Accessing the System Catalog. 1-12
Updating System Catalog Data 1-13
Structure of the System Catalog 1-14
SYSAGGREGATES 1-17
SYSAMS 1-18
SYSATTRTYPES 1-21
SYSBLOBS 1-22

SYSCASTS 1-23
SYSCHECKS 1-24
SYSCHECKUDRDEP 1-25
SYSCOLATTRIBS 1-26
SYSCOLAUTH 1-27
SYSCOLDEPEND 1-28
SYSCOLUMNS 1-29
SYSCONSTRAINTS 1-34
SYSDEFAULTS 1-35
SYSDEPEND 1-37
SYSDISTRIB 1-37
SYSDOMAINS 1-39
SYSERRORS 1-39
SYSEXTCOLS 1-40
SYSEXTDFILES 1-41
SYSEXTERNAL 1-41
SYSFRAGAUTH 1-42
SYSFRAGMENTS 1-43
SYSINDEXES 1-46
SYSINDICES 1-48
SYSINHERITS. 1-50
SYSLANGAUTH. 1-50
SYSLOGMAP 1-51
SYSNEWDEPEND 1-51
SYSOBJSTATE. 1-52
SYSOPCLASSES 1-53
SYSOPCLSTR 1-54
SYSPROCAUTH 1-56
SYSPROCBODY 1-57
SYSPROCEDURES 1-58
SYSPROCPLAN 1-62
SYSREFERENCES 1-63
SYSREPOSITORY 1-64
SYSROLEAUTH 1-65
SYSROUTINELANGS 1-65
SYSSYNONYMS 1-66
SYSSYNTABLE 1-66
SYSTABAMDATA 1-67
SYSTABAUTH 1-68

SYSTABLES 1-69
SYSTRACECLASSES 1-72
SYSTRACEMSGS 1-73
SYSTRIGBODY 1-74
SYSTRIGGERS 1-75
SYSUSERS 1-76
SYSVIEWS 1-76
SYSVIOLATIONS 1-77
SYSXTDDESC 1-78
SYSXTDTYPEAUTH 1-78
SYSXTDTYPES 1-79
Information Schema 1-80
Generating the Information Schema Views 1-81
Accessing the Information Schema Views 1-81
Structure of the Information Schema Views 1-82

Chapter 2 Data Types

In This Chapter 2-5
Summary of Data Types 2-5
Description of Data Types 2-9
BLOB 2-9
BOOLEAN 2-10
BYTE 2-11
CHAR(n) 2-12
CHARACTER(n) 2-14
CHARACTER VARYING(m,r) 2-14
CLOB 2-14
DATE 2-16
DATETIME 2-17
DEC. 2-21
DECIMAL 2-21
Distinct 2-23
DOUBLE PRECISION 2-24
FLOAT(n) 2-24
INT 2-25
INT8 2-25
INTEGER 2-26
INTERVAL 2-26
LIST(e) 2-29

LVARCHAR 2-31
MONEY(p,s) 2-31
MULTISET(e) 2-32
NCHAR(n) 2-34
NUMERIC(p,s) 2-34
NVARCHAR(m,r) 2-34
Opaque 2-34
REAL 2-35
Row, Named 2-35
Row, Unnamed 2-37
SERIAL(n) 2-39
SERIAL8. 2-40
SET(e) 2-41
SMALLFLOAT 2-43
SMALLINT. 2-43
TEXT 2-44
VARCHAR(m,r) 2-46
Built-In Data Types 2-48
Large-Object Data Types 2-48
Time Data Types 2-51
Extended Data Types 2-57
Complex Data Types 2-58
DISTINCT Data Types 2-61
OPAQUE Data Types 2-61
Data Type Casting and Conversion 2-62
Using Built-in Casts 2-63
Using User-Defined Casts 2-66
Determining Which Cast to Apply 2-67
Casts for Distinct Types 2-68
What Extended Data Types Can Be Cast? 2-69
Operator Precedence 2-70

Chapter 3 Environment Variables

In This Chapter 3-5
Types of Environment Variables 3-5
Where to Set Environment Variables on UNIX 3-6
Where to Set Environment Variables on Windows 3-7
Using Environment Variables on UNIX 3-8
Setting Environment Variables in a Configuration File . . . 3-8
Setting Environment Variables at Login Time 3-9
Syntax for Setting Environment Variables. 3-9
Unsetting Environment Variables 3-10
Modifying an Environment-Variable Setting 3-10
Viewing Your Environment-Variable Settings 3-11
Checking Environment Variables with the chkenv Utility . . 3-11
Rules of Precedence 3-12
Using Environment Variables on Windows 3-13
Setting Environment Variables for Native Windows Applications 3-13
Setting Environment Variables for Command-Prompt Utilities 3-15
Rules of Precedence 3-18
List of Environment Variables 3-18
Environment Variables 3-23
AC_CONFIG 3-24
AFDEBUG 3-24
CPFIRST 3-25
DBACCNOIGN. 3-26
DBANSIWARN 3-27
DBBLOBBUF 3-28
DBCENTURY 3-28
DBDATE 3-32
DBDELIMITER 3-35
DBEDIT 3-35
DBFLTMASK 3-36
DBLANG 3-37
DBMONEY 3-38
DBNLS. 3-40
DBONPLOAD 3-41
DBPATH 3-42
DBPRINT 3-44
DBREMOTECMD 3-45

DBSPACETEMP 3-46
DBTEMP 3-48
DBTIME 3-48
DBUPSPACE 3-52
DEFAULT_ATTACH 3-53
DELIMIDENT 3-53
ENVIGNORE 3-54
FET_BUF_SIZE 3-55
IFMX_SMLTBL_BROADCAST_SIZE 3-56
IFX_DEF_TABLE_LOCKMODE 3-56
IFX_DIRECTIVES 3-58
IFX_LONGID 3-59
IFX_NETBUF_PVTPOOL_SIZE 3-60
IFX_NETBUF_SIZE 3-60
IFX_UPDDESC 3-61
IMCADMIN 3-61
IMCCONFIG 3-62
IMCSERVER 3-62
INFORMIXC 3-63
INFORMIXCONCSMCFG 3-63
INFORMIXCONRETRY 3-64
INFORMIXCONTIME 3-65
INFORMIXCPPMAP 3-66
INFORMIXDIR 3-67
INFORMIXKEYTAB. 3-67
INFORMIXOPCACHE 3-68
INFORMIXSERVER 3-68
INFORMIXSHMBASE 3-69
INFORMIXSQLHOSTS. 3-70
INFORMIXSTACKSIZE 3-71
INFORMIXTERM 3-72
INF_ROLE_SEP 3-73
INTERACTIVE_DESKTOP_OFF 3-74
ISM_COMPRESSION 3-74
ISM_DEBUG_FILE 3-75
ISM_DEBUG_LEVEL 3-75
ISM_ENCRYPTION 3-76
ISM_MAXLOGSIZE. 3-76
ISM_MAXLOGVERS 3-77

JAR_TEMP_PATH 3-77
JAVA_COMPILER 3-78
JVM_MAX_HEAP_SIZE. 3-78
LD_LIBRARY_PATH 3-79
LIBPATH 3-79
NODEFDAC 3-80
ONCONFIG 3-80
OPTCOMPIND 3-81
OPTMSG 3-82
OPTOFC 3-83
OPT_GOAL 3-83
PATH 3-84
PDQPRIORITY 3-85
PLCONFIG 3-87
PLOAD_LO_PATH 3-87
PLOAD_SHMBASE 3-88
PSORT_DBTEMP 3-89
PSORT_NPROCS 3-89
SHLIB_PATH 3-91
STMT_CACHE 3-92
TERM 3-92
TERMCAP 3-93
TERMINFO 3-94
THREADLIB. 3-94
XFER_CONFIG 3-95
Index of Environment Variables 3-95

Appendix A The stores_demo Database

Appendix B The sales_demo and superstores_demo Databases

Glossary

Index

Introduction

In This Introduction 3

About This Manual. 3
Types of Users 4
Software Dependencies 4
Assumptions About Your Locale. 5
Demonstration Database 5

New Features in Dynamic Server, Version 9.3 6
SQL Enhancements 6

Documentation Conventions 7
Typographical Conventions 7
Icon Conventions 8
Comment Icons 8
Feature, Product, and Platform Icons 8
Compliance Icons 9
Syntax Conventions 10
Sample-Code Conventions 11

Additional Documentation 12

Related Reading 14

Compliance with Industry Standards 15

Informix Welcomes Your Comments 15

In This Introduction

This introduction provides an overview of the information in this manual and describes the conventions it uses.

About This Manual

This manual includes information about system catalog tables, data types, and environment variables that Informix products use. It also includes a glossary that contains definitions of common terms found in Informix documentation and a description of the demonstration databases that Version 9.3 of Informix Dynamic Server 2000 and Version 8.3 of Informix Extended Parallel Server provide.

This manual is one of a series of manuals that discusses the Informix implementation of SQL. The *Informix Guide to SQL: Syntax* contains all the syntax descriptions for SQL and stored procedure language (SPL). The *Informix Guide to SQL: Tutorial* shows how to use basic and advanced SQL and SPL routines to access and manipulate the data in your databases. The *Informix Guide to Database Design and Implementation* shows how to use SQL to implement and manage your databases.

Types of Users

This manual is written for the following users:

- Database users
- Database administrators
- Database server administrators
- Database-application programmers
- Performance engineers

This manual assumes that you have the following background:

- A working knowledge of your computer, your operating system, and the utilities that your operating system provides
- Some experience working with relational databases or exposure to database concepts
- Some experience with computer programming
- Some experience with database server administration, operating-system administration, or network administration

If you have limited experience with relational databases, SQL, or your operating system, refer to the *Getting Started* manual for your database server for a list of supplementary titles.

Software Dependencies

This manual is written with the assumption that you are using one of the following database servers:

- Informix Extended Parallel Server, Version 8.3
- Informix Dynamic Server, Version 9.3

Assumptions About Your Locale

Informix products can support many languages, cultures, and code sets. All the information related to character set, collation, and representation of numeric data, currency, date, and time is brought together in a single environment, called a Global Language Support (GLS) locale.

This manual assumes that you use the U.S. 8859-1 English locale as the default locale. The default is **en_us.8859-1** (ISO 8859-1) on UNIX platforms or **en_us.1252** (Microsoft 1252) for Windows NT environments. This locale supports U.S. English format conventions for dates, times, and currency, and also supports the ISO 8859-1 or Microsoft 1252 code set, which includes the ASCII code set plus many 8-bit characters such as é, è, and ñ.

If you plan to use nondefault characters in your data or your SQL identifiers, or if you want to conform to the nondefault collation rules of character data, you need to specify the appropriate nondefault locale.

For instructions on how to specify a nondefault locale, additional syntax, and other considerations related to GLS locales, see the *Informix Guide to GLS Functionality*.

Demonstration Database

The DB-Access utility, which Informix provides with its database server products, includes one or more of the following demonstration databases:

- The **stores_demo** database illustrates a relational schema with information about a fictitious wholesale sporting-goods distributor. Many examples in Informix manuals are based on the **stores_demo** database.

XPS

- The **sales_demo** database illustrates a dimensional schema for data-warehousing applications. For conceptual information about dimensional data modeling, see the *Informix Guide to Database Design and Implementation*. ♦

IDS

- The **superstores_demo** database illustrates an object-relational schema. The **superstores_demo** database contains examples of extended data types, type and table inheritance, and user-defined routines. ♦

For information about how to create and populate the demonstration databases, see the *DB-Access User's Manual*. For descriptions of the databases and their contents, see the *Informix Guide to SQL: Reference*.

The scripts that you use to install the demonstration databases reside in the **$INFORMIXDIR/bin** directory on UNIX platforms and in the **%INFORMIXDIR%\bin** directory in Windows environments.

New Features in Dynamic Server, Version 9.3

The following table provides information about the new features for Informix Dynamic Server, Version 9.3, that this manual covers. To go to the desired page, click a blue hyperlink. For a description of all new features, see the *Getting Started* manual.

SQL Enhancements

Version 9.3 includes an environment variable that allows the user or the DBA to specify default locking modes for the database.

New Features	Reference
Configurable default lock modes	"IFX_DEF_TABLE_LOCKMODE" on page 3-56

Documentation Conventions

This section describes the conventions that this manual uses. These conventions make it easier to gather information from this and other volumes in the documentation set.

Typographical Conventions

This manual uses the following conventions to introduce new terms, illustrate screen displays, describe command syntax, and so forth.

Convention	Meaning
KEYWORD	All primary elements in a programming language statement (keywords) appear in uppercase letters in a serif font.
italics ***italics*** *`italics`*	Within text, new terms and emphasized words appear in italics. Within syntax and code examples, variable values that you are to specify appear in italics.
boldface ***boldface***	Names of program entities (such as classes, events, and tables), environment variables, file and pathnames, and interface elements (such as icons, menu items, and buttons) appear in boldface.
`monospace` *`monospace`*	Information that the product displays and information that you enter appear in a monospace typeface.
KEYSTROKE	Keys that you are to press appear in uppercase letters in a sans serif font.
♦	This symbol indicates the end of product- or platform-specific information.
→	This symbol indicates a menu item. For example, "Choose **Tools→Options**" means choose the **Options** item from the **Tools** menu.

Tip: *When you are instructed to "enter" characters or to "execute" a command, immediately press* RETURN *after the entry. When you are instructed to "type" the text or to "press" other keys, no* RETURN *is required.*

Icon Conventions

Throughout the documentation, you will find text that is identified by several different types of icons. This section describes these icons.

Comment Icons

Comment icons identify three types of information, as the following table describes. This information always appears in italics.

Icon	Label	Description
	Warning:	Identifies paragraphs that contain vital instructions, cautions, or critical information
	Important:	Identifies paragraphs that contain significant information about the feature or operation that is being described
	Tip:	Identifies paragraphs that offer additional details or shortcuts for the functionality that is being described

Feature, Product, and Platform Icons

Feature, product, and platform icons identify paragraphs that contain feature-specific, product-specific, or platform-specific information.

Icon	Description
C	Identifies information that is specific to C user-defined routines (UDRs)
DB	Identifies information that is specific to DB-Access
E/C	Identifies information that is specific to Informix ESQL/C

(1 of 2)

Icon	Description
Ext	Identifies information that is specific to external routines, that is, UDRs written in either C or Java language
GLS	Identifies information that relates to the Informix Global Language Support (GLS) feature
IDS	Identifies information or syntax that is specific to Informix Dynamic Server
Java	Identifies information that is specific to UDRs written in Java
UNIX	Identifies information that is specific to the UNIX operating system
Windows	Identifies information that applies to all Windows environments
XPS	Identifies information or syntax that is specific to Informix Extended Parallel Server

(2 of 2)

These icons can apply to a row in a table, one or more paragraphs, or an entire section. A ♦ symbol indicates the end of the feature-specific, product-specific, or platform-specific information.

Compliance Icons

Compliance icons indicate paragraphs that provide guidelines for complying with a standard.

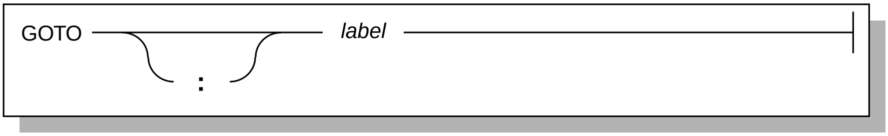
Icon	Description
ANSI	Identifies information that is specific to an ANSI-compliant database
X/O	Identifies functionality that conforms to X/Open
+	Identifies information that is an Informix extension to ANSI SQL-92 entry-level standard SQL

These icons can apply to an entire section or to one or more paragraphs within a section. If an icon appears next to a section heading, the compliance information ends at the next heading at the same or higher level. A ♦ symbol indicates the end of compliance information that appears in one or more paragraphs within a section.

Syntax Conventions

This section describes conventions for syntax diagrams. Each diagram displays the sequences of required and optional keywords, terms, and symbols that are valid in a given statement, command line, or other specification, as in Figure 1.

Figure 1
Example of a Simple Syntax Diagram



Keep in mind the following rules when you read syntax diagrams in this book:

- To make keywords (like GOTO in Figure 1) easy to identify, they are shown in UPPERCASE letters, even though you can type them in either uppercase or lowercase letters.
- Terms for which you must supply specific values are in *italics*. In Figure 1, you must replace *label* with an identifier. Below each diagram that contains an italicized term, a table explains what you can substitute for the term.
- All the punctuation and other nonalphabetic characters are literal symbols. In Figure 1, the colon is a literal symbol.
- Each syntax diagram begins at the upper-left corner and ends at the upper-right corner with a vertical terminator. Between these points, any path that does not stop or reverse direction describes a possible form of the statement.

Syntax elements in a path represent terms, keywords, symbols, and segments that can appear in your statement. The path always approaches elements from the left and continues to the right, except in the case of separators in loops. For separators in loops, the path approaches counterclockwise from the right. Unless otherwise noted, at least one blank character separates syntax elements.

Sample-Code Conventions

Examples of SQL code occur throughout this manual. Except where noted, the code is not specific to any single Informix application development tool. If only SQL statements are listed in the example, they are not delimited by semicolons. For instance, you might see the code in the following example:

```
CONNECT TO stores_demo
...

DELETE FROM customer
   WHERE customer_num = 121
...

COMMIT WORK
DISCONNECT CURRENT
```

To use this SQL code for a specific product, you must apply the syntax rules for that product. For example, if you are using DB-Access, you must delimit multiple statements with semicolons. If you are using an SQL API, you must use EXEC SQL at the start of each statement and a semicolon (or other appropriate delimiter) at the end of the statement.

Tip: *Ellipsis points in a code example indicate that more code would be added in a full application, but it is not necessary to show it to describe the concept being discussed.*

For detailed directions on using SQL statements for a particular application development tool or SQL API, see the manual for your product.

Additional Documentation

Informix Dynamic Server documentation is provided in a variety of formats:

- **Online manuals.** The Informix OnLine Documentation Web site at http://www.informix.com/answers contains manuals that Informix provides for your use. This Web site enables you to print chapters or entire books.
- **Online help.** Informix provides online help with each graphical user interface (GUI) that displays information about those interfaces and the functions that they perform. Use the help facilities that each GUI provides to display the online help.

  This facility can provide context-sensitive help, an error message reference, language syntax, and more. To order a printed manual, call 1-800-331-1763 or send email to moreinfo@informix.com. Provide the following information when you place your order:

  - The documentation that you need
  - The quantity that you need
  - Your name, address, and telephone number
- **Documentation notes.** Documentation notes, which contain additions and corrections to the manuals, are also located at the OnLine Documentation site at http://www.informix.com/answers. Examine these files before you begin using your database server.

- **Release notes.** Release notes contain vital information about application and performance issues. These files are located at http://www.informix.com/informix/services/techinfo. This site is a password controlled site. Examine these files before you begin using your database server.

  Documentation notes, release notes, and machine notes are also located in the directory where the product is installed. The following table describes these files.

UNIX

On UNIX platforms, the following online files appear in the **$INFORMIXDIR/release/en_us/0333** directory.

Online File	Purpose
sqlr_docnotes_9.30.html	The documentation notes file for your version of this manual describes topics that are not covered in the manual or that were modified since publication.
release_notes_9.30.html	The release notes file describes feature differences from earlier versions of Informix products and how these differences might affect current products. This file also contains information about any known problems and their workarounds.
machine_notes_9.30.txt	The machine notes file describes any special actions that you must take to configure and use Informix products on your computer. Machine notes are named for the product described.

♦

Windows

The following items appear in the **Informix** folder. To display this folder, choose **Start→Programs→Informix Dynamic Server 9.30→ Documentation Notes or Release Notes** from the task bar.

Program Group Item	Description
Documentation Notes	This item includes additions or corrections to manuals with information about features that might not be covered in the manuals or that have been modified since publication.
Release Notes	This item describes feature differences from earlier versions of Informix products and how these differences might affect current products. This file also contains information about any known problems and their workarounds.

Machine notes do not apply to Windows NT platforms. ♦

- **Error message files**. Informix software products provide ASCII files that contain Informix error messages and their corrective actions. For a detailed description of these error messages, refer to *Informix Error Messages* in Answers OnLine.

UNIX

To read the error messages on UNIX, use the following command.

Command	Description
finderr	Displays error messages online

♦

WIN NT

To read error messages and corrective actions on Windows NT, use the **Informix Find Error** utility. To display this utility, choose **Start→Programs→Informix** from the task bar. ♦

Related Reading

For a list of publications that provide an introduction to database servers and operating-system platforms, refer to your *Getting Started* manual.

Compliance with Industry Standards

The American National Standards Institute (ANSI) has established a set of industry standards for SQL. Informix SQL-based products are fully compliant with SQL-92 Entry Level (published as ANSI X3.135-1992), which is identical to ISO 9075:1992. In addition, many features of Informix database servers comply with the SQL-92 Intermediate and Full Level and X/Open SQL CAE (common applications environment) standards.

Informix Welcomes Your Comments

We want to know about any corrections or clarifications that you would find useful in our manuals that would help us with future versions. Include the following information:

- The name and version of the manual that you are using
- Any comments that you have about the manual
- Your name, address, and phone number

Send electronic mail to us at the following address:

doc@informix.com

This address is reserved for reporting errors and omissions in our documentation. For immediate help with a technical problem, contact Informix Customer Services.

We appreciate your suggestions.

Chapter

1

System Catalog Tables

In This Chapter 1-5

Objects That the System Catalog Tables Track. 1-5

Using the System Catalog 1-6
Accessing the System Catalog. 1-12
Updating System Catalog Data 1-13

Structure of the System Catalog 1-14
SYSAGGREGATES 1-17
SYSAMS 1-18
SYSATTRTYPES 1-21
SYSBLOBS 1-22
SYSCASTS 1-23
SYSCHECKS 1-24
SYSCHECKUDRDEP. 1-25
SYSCOLATTRIBS 1-26
SYSCOLAUTH 1-27
SYSCOLDEPEND 1-28
SYSCOLUMNS. 1-29
SYSCONSTRAINTS 1-34
SYSDEFAULTS 1-35
SYSDEPEND 1-37
SYSDISTRIB 1-37
SYSDOMAINS 1-39
SYSERRORS. 1-39
SYSEXTCOLS 1-40
SYSEXTDFILES. 1-41
SYSEXTERNAL 1-41

SYSFRAGAUTH 1-42
SYSFRAGMENTS 1-43
SYSINDEXES 1-46
SYSINDICES. 1-48
SYSINHERITS 1-50
SYSLANGAUTH 1-50
SYSLOGMAP 1-51
SYSNEWDEPEND. 1-51
SYSOBJSTATE 1-52
SYSOPCLASSES 1-53
SYSOPCLSTR 1-54
SYSPROCAUTH 1-56
SYSPROCBODY 1-57
SYSPROCEDURES. 1-58
SYSPROCPLAN 1-62
SYSREFERENCES 1-63
SYSREPOSITORY 1-64
SYSROLEAUTH 1-65
SYSROUTINELANGS 1-65
SYSSYNONYMS 1-66
SYSSYNTABLE 1-66
SYSTABAMDATA 1-67
SYSTABAUTH 1-68
SYSTABLES 1-69
SYSTRACECLASSES 1-72
SYSTRACEMSGS 1-73
SYSTRIGBODY 1-74
SYSTRIGGERS 1-75
SYSUSERS 1-76
SYSVIEWS 1-76
SYSVIOLATIONS 1-77
SYSXTDDESC 1-78
SYSXTDTYPEAUTH 1-78
SYSXTDTYPES 1-79

Information Schema 1-80
Generating the Information Schema Views 1-81
Accessing the Information Schema Views 1-81
Structure of the Information Schema Views. 1-82
The tables Information Schema View 1-82
The columns Information Schema View 1-83
The sql_languages Information Schema View 1-84
The server_info Information Schema View. 1-85

In This Chapter

The system catalog consists of tables that describe the structure of the database. Sometimes called the “data dictionary,” these tables contain everything that the database knows about itself. Each system catalog table contains specific information about elements in the database.

This chapter provides information about the structure, contents, and use of the system catalog tables. It also discusses the Information Schema, which provides information about the tables, views, and columns on the current database server.

Objects That the System Catalog Tables Track

The system catalog tables track the following categories of objects:

- Tables, views, sequences, and synonyms
- Columns, constraints, and indexes
- Triggers
- Procedures, functions, routines and associated messages
- Authorized users and privileges associated with database objects

IDS

- Data types and casts
- Aggregate functions
- Access methods and operator classes
- Inheritance relationships ♦

Using the System Catalog

Informix database servers automatically generate the system catalog tables when you create a database. You can query them as you would query any other table in the database. The system catalog tables for a newly created database reside in a common area of the disk called a *dbspace*. Every database has its own system catalog tables. All tables in the system catalog have the prefix **sys** (for example, the **systables** system catalog table).

Not all tables with the prefix ***sys*** are true system catalog tables. For example, a common DataBlade module builds a **sysbuiltintypes** table that looks like a system table and contains similar data, and the **syscdr** database supports the Enterprise Replication feature. These tables, however, have a **tabid** > 99. System catalog tables all have a **tabid** < 99. (See later in this section and "SYSTABLES" on page 1-69 for more information about **tabid** numbers that the database assigns to tables.)

Tip: *Do not confuse the system catalog tables of a database with the tables in the **sysmaster**, **sysutils**, or **sysuuid** databases. The names of tables in those databases also have the **sys** prefix, but they contain information about an entire database server, which might manage multiple databases. Information in the **sysmaster**, **sysutils**, and **sysuuid** tables is primarily useful for database system administrators (DBSAs). For more information about these databases and their tables, see the "Administrator's Guide" and "Administrator's Reference."*

The database server accesses the system catalog constantly. Each time an SQL statement is processed, the database server accesses the system catalog to determine system privileges, add or verify table or column names, and so on. For example, the following CREATE SCHEMA block adds the **customer** table, with its respective indexes and privileges, to the **stores_demo** database. This block also adds a view, **california**, that restricts the *view* of the **customer** table to only the first and last names of the customer, the company name, and the telephone number for all customers who reside in California.

```
CREATE SCHEMA AUTHORIZATION maryl
CREATE TABLE customer
   (customer_num SERIAL(101), fname CHAR(15), lname CHAR(15), company
CHAR(20),
   address1 CHAR(20), address2 CHAR(20), city CHAR(15), state CHAR(2),
   zipcode CHAR(5), phone CHAR(18))
GRANT ALTER, ALL ON customer TO cathl WITH GRANT OPTION AS maryl
```

```
GRANT SELECT ON customer TO public
GRANT UPDATE (fname, lname, phone) ON customer TO nhowe
CREATE VIEW california AS
   SELECT fname, lname, company, phone FROM customer WHERE state = 'CA'
CREATE UNIQUE INDEX c_num_ix ON customer (customer_num)
CREATE INDEX state_ix ON customer (state)
```

To process this CREATE SCHEMA block, the database server first accesses the system catalog to verify the following information:

- The new table and view names do not already exist in the database. (If the database is ANSI-compliant, the database server verifies that the new names do not already exist for the specified owners.)
- The user has permission to create tables and grant user privileges.
- The column names in the CREATE VIEW and CREATE INDEX statements exist in the **customer** table.

In addition to verifying this information and creating two new tables, the database server adds new rows to the following system catalog tables:

- **systables**
- **syscolumns**
- **sysviews**
- **systabauth**
- **syscolauth**
- **sysindexes**
- IDS **sysindices** ♦

The following two new rows of information are added to the **systables** system catalog table after the CREATE SCHEMA block is run.

	First row	Second row
tabname	customer	california
owner	maryl	maryl
partnum	16778361	0
tabid	101	102
rowsize	134	134

(1 of 2)

	First row	Second row
ncols	10	4
nindexes	2	0
nrows	0	0
created	01/26/1999	01/26/1999
version	1	0
tabtype	T	V
locklevel	P	B
npused	0	0
fextsize	16	0
nextsize	16	0
flags	0	0
site		
dbname		

(2 of 2)

Each table recorded in the **systables** system catalog table is assigned a **tabid**, a system-assigned sequential ID number that uniquely identifies each table in the database. The system catalog tables receive 2-digit **tabid** numbers, and the user-created tables receive sequential **tabid** numbers that begin with `100`.

The CREATE SCHEMA block adds 14 rows to the **syscolumns** system catalog table. These rows correspond to the columns in the table **customer** and the view **california**, as the following example shows.

colname	tabid	colno	coltype	collength	colmin	colmax
customer_num	101	1	262	4		
fname	101	2	0	15		
lname	101	3	0	15		

(1 of 2)

colname	tabid	colno	coltype	collength	colmin	colmax
company	101	4	0	20		
address1	101	5	0	20		
address2	101	6	0	20		
city	101	7	0	15		
state	101	8	0	2		
zipcode	101	9	0	5		
phone	101	10	0	18		
fname	102	1	0	15		
lname	102	2	0	15		
company	102	3	0	20		
phone	102	4	0	18		

(2 of 2)

In the **syscolumns** table, each column within a table is assigned a sequential column number, **colno**, that uniquely identifies the column within its table. In the **colno** column, the **fname** column of the **customer** table is assigned the value 2 and the **fname** column of the view **california** is assigned the value 1.

The **colmin** and **colmax** columns contain no entries. These two columns contain values when a column is the first key (or the only key) in an index, and has no NULL or duplicate values, and the UPDATE STATISTICS statement has been run.

The database server also adds rows to the **sysviews** system catalog table. The **sysviews** table contains the CREATE VIEW statement that creates the view. Each line of the CREATE VIEW statement in the current schema is stored in this table. In the **viewtext** column, the **x0** that precedes the column names in the statement (for example, **x0.fname**) operates as an alias name that distinguishes among the same columns that are used in a self-join.

The CREATE SCHEMA block also adds rows to the **systabauth** system catalog table. These rows correspond to the user privileges granted on **customer** and **california** tables, as the following example shows.

grantor	grantee	tabid	tabauth
maryl	public	101	su-idx--
maryl	cathl	101	SU-IDXAR
maryl	nhowe	101	--*-----
	maryl	102	SU-ID---

The **tabauth** column specifies the table-level privileges granted to users on the **customer** and **california** tables. This column uses an 8-byte pattern, such as `s` (Select), `u` (Update), `*` (column-level privilege), `i` (Insert), `d` (Delete), `x` (Index), `a` (Alter), and `r` (References), to identify the type of privilege. In this example, the user **nhowe** has column-level privileges on the **customer** table. Where a hyphen (`-`) appears, the user has not been granted the privilege whose position the hyphen occupies within the **tabauth** value.

If the **tabauth** privilege code appears in uppercase (for example, `S` for Select), the user has this privilege and can also grant it to others; but if the privilege code is lowercase (for example, `s` for Select), the user cannot grant it to others.

In addition, three rows are added to the **syscolauth** system catalog table. These rows correspond to the user privileges that are granted on specific columns in the **customer** table, as the following example shows.

grantor	grantee	tabid	colno	colauth
maryl	nhowe	101	2	-u-
maryl	nhowe	101	3	-u-
maryl	nhowe	101	10	-u-

The **colauth** column specifies the column-level privileges that are granted on the **customer** table. This column uses a 3-byte pattern, such as `s` (Select), `u` (Update), and `r` (References), to identify the type of privilege. For example, the user **nhowe** has Update privileges on the second column (because the **colno** value is `2`) of the **customer** table (indicated by **tabid** value of `101`).

The CREATE SCHEMA block adds two rows to the **sysindexes** system catalog table (the **sysindices** table for Dynamic Server). These rows correspond to the indexes created on the **customer** table, as the following example shows.

idxname	c_num_ix	state_ix
owner	maryl	maryl
tabid	101	101
idxtype	U	D
clustered		
part1	1	8
part2	0	0
part3	0	0
part4	0	0
part5	0	0
part6	0	0
part7	0	0
part8	0	0
part9	0	0
part10	0	0
part11	0	0
part12	0	0
part13	0	0
part14	0	0

(1 of 2)

idxname	c_num_ix	state_ix
part15	0	0
part16	0	0
levels		
leaves		
nunique		
clust		

(2 of 2)

In this table, the **idxtype** column identifies whether the created index requires unique values (U) or accepts duplicate values (D). For example, the **c_num_ix** index on the **customer.customer_num** column is unique.

Accessing the System Catalog

Normal user access to the system catalog is read-only. Users with Connect or Resource privileges cannot alter the catalog, but they can access data in the system catalog tables on a read-only basis using standard SELECT statements. For example, the following SELECT statement displays all the table names and corresponding **tabid** codes of user-created tables in the database:

```
SELECT tabname, tabid FROM systables WHERE tabid > 99
```

When you use DB-Access, only the tables that you created are displayed. To display the system catalog tables, enter the following statement:

```
SELECT tabname, tabid FROM systables WHERE tabid < 100
```

You can use SUBSTR or SUBSTRING function to select only part of a source string. To display the list of tables in columns, enter the following statement:

```
SELECT SUBSTR(tabname, 1, 18), tabid FROM systables
```

Warning: *Although user **informix** and DBAs can modify most system catalog tables (only user **informix** can modify **systables**), Informix strongly recommends that you do not update, delete, or insert any rows in them. Modifying the system catalog tables can destroy the integrity of the database. Informix does support use of the ALTER TABLE statement to modify the size of the next extent of system catalog tables.*

IDS

In certain catalog tables of Dynamic Server, however, it is valid to add entries to the system catalog tables. For instance, in the case of the ***syserrors*** *system catalog table and the* ***systracemsgs*** *system catalog table, a developer of DataBlade modules can directly insert message entries that appear in these system catalog tables.* ♦

Updating System Catalog Data

The optimizer in Informix database servers determines the most efficient strategy for executing SQL queries. The optimizer allows you to query the database without having to consider fully which tables to search first in a join or which indexes to use. The optimizer uses information from the system catalog to determine the best query strategy.

If you use the UPDATE STATISTICS statement to update the system catalog, you can ensure that the information provided to the optimizer is current. When you delete or modify a table, the database server does not automatically update the related statistical data in the system catalog. For example, if you delete one or more rows in a table with the DELETE statement, the **nrows** column in the **systables** system catalog table, which holds the number of rows for that table, is not updated automatically.

The UPDATE STATISTICS statement causes the database server to recalculate data in the **systables**, **sysdistrib**, **syscolumns**, and **sysindexes** (**sysindices** for Dynamic Server) system catalog tables. After you run UPDATE STATISTICS, the **systables** system catalog table holds the correct value in the **nrows** column. If you specify MEDIUM or HIGH mode with the UPDATE STATISTICS statement, the **sysdistrib** system catalog table holds the updated data-distribution data after you run UPDATE STATISTICS.

Whenever you modify a table extensively, use the UPDATE STATISTICS statement to update data in the system catalog. For more information on the UPDATE STATISTICS statement, see the *Informix Guide to SQL: Syntax*.

Structure of the System Catalog

The following system catalog tables describe the structure of an Informix database. Here ✔ indicates that IDS (or XPS) supports the table.

System Catalog Table	XPS	IDS	Page
sysaggregates		✔	1-17
sysams		✔	1-18
sysattrtypes		✔	1-21
sysblobs	✔	✔	1-22
syscasts		✔	1-23
syschecks	✔	✔	1-24
syscheckudrdep		✔	1-25
syscolattribs		✔	1-26
syscolauth	✔	✔	1-27
syscoldepend	✔	✔	1-28
syscolumns	✔	✔	1-29
sysconstraints	✔	✔	1-34
sysdefaults	✔	✔	1-35
sysdepend	✔	✔	1-37
sysdistrib	✔	✔	1-37
sysdomains		✔	1-39
syserrors		✔	1-39
sysextcols	✔		1-40
sysextdfiles	✔		1-41
sysexternal	✔		1-41

(1 of 3)

System Catalog Table	XPS	IDS	Page
sysfragauth		✔	1-42
sysfragments	✔	✔	1-43
sysindexes	✔	✔	1-46
sysindices		✔	1-48
sysinherits		✔	1-50
syslangauth		✔	1-50
syslogmap		✔	1-51
sysnewdepend	✔		1-51
sysobjstate		✔	1-52
sysopclasses		✔	1-53
sysopclstr	✔	✔	1-54
sysprocauth	✔	✔	1-56
sysprocbody	✔	✔	1-57
sysprocedures	✔	✔	1-58
sysprocplan	✔	✔	1-62
sysreferences	✔	✔	1-63
sysrepository	✔		1-64
sysroleauth		✔	1-65
sysroutinelangs		✔	1-65
syssynonyms	✔	✔	1-66
syssyntable	✔	✔	1-66
systabamdata		✔	1-67
systabauth	✔	✔	1-68
systables	✔	✔	1-69

(2 of 3)

System Catalog Table	XPS	IDS	Page
systraceclasses		✔	1-72
systracemsgs		✔	1-73
systrigbody	✔	✔	1-74
systriggers	✔	✔	1-75
sysusers	✔	✔	1-76
sysviews	✔	✔	1-76
sysviolations	✔	✔	1-77
sysxtddesc		✔	1-78
systdtypeauth		✔	1-78
sysxtdtypes		✔	1-79

(3 of 3)

GLS

If the database locale is the default (U. S. English, ISO 8859-1 codeset), then the character column types are CHAR and VARCHAR. For all other locales and collation orders, the character column types are national character types, NCHAR and NVARCHAR. For more information about locales and collation order of data types, see the *Informix Guide to GLS Functionality*. For more information about SQL data types, see Chapter 2 of this manual. ♦

IDS

SYSAGGREGATES

The **sysaggregates** system catalog table records user-defined aggregates (UDAs). The **sysaggregates** table has the following columns.

Column	Type	Explanation
name	VARCHAR(128)	Name of the aggregate
owner	CHAR(32)	Name of the owner of the aggregate
aggid	SERIAL	Unique code identifying the aggregate
init_func	VARCHAR(128)	Name of initialization UDR
iter_func	VARCHAR(128)	Name of iterator UDR
combine_func	VARCHAR(128)	Name of combine UDR
final_func	VARCHAR(128)	Name of finalization UDR
handlesnulls	BOOLEAN	NULL handling indicator: `t` = handles NULLs `f` = does not handle NULLs

Each user-defined aggregate has one entry in **sysaggregates** that is uniquely identified by its identifying code (the **aggid** value). Only user-defined aggregates (aggregates that are not built in) have entries in **sysaggregates**.

Both a simple index on the **aggid** column and a composite index on the **name** and **owner** columns require unique values.

IDS

SYSAMS

The **sysams** system catalog table contains information that is needed to use built-in access methods as well as those created by the CREATE ACCESS METHOD statement of SQL that is described in the *Informix Guide to SQL: Syntax*. The **sysams** table has the following columns.

Column	Type	Explanation
am_name	VARCHAR(128)	Name of the access method
am_owner	CHAR(32)	Name of the owner of the access method
am_id	INTEGER	Unique identifying code for an access method. This corresponds to the **am_id** columns in the **systables**, **sysindices**, and **sysopclasses** tables.
am_type	CHAR(1)	Type of access method: P = Primary S = Secondary
am_sptype	CHAR(3)	Types of spaces in which the access method can exist: A *or* a = all types: extspaces, dbspaces, and sbspaces. If the access method is not user-defined (that is, if it is built-in or registered during database creation by the server), it supports dbspaces. D *or* d = dbspaces only S *or* s = sbspaces only (smart-large-object space) X *or* x = extspaces only
am_defopclass	INTEGER	Unique identifying code for default-operator class. Value is the **opclassid** from the entry for this operator class in the **sysopclasses** table.

(1 of 3)

Column	Type	Explanation
am_keyscan	INTEGER	Whether a secondary access method supports a key scan. (An access method supports a key scan if it can return a key as well as a rowid from a call to the **am_getnext** function.) (`0` = FALSE; `Non-Zero` = TRUE)
am_unique	INTEGER	Whether a secondary access method can support unique keys (`0` = FALSE; `Non-Zero` = TRUE)
am_cluster	INTEGER	Whether a primary access method supports clustering (`0` = FALSE; `Non-Zero` = TRUE)
am_rowids	INTEGER	Whether a primary access method supports rowids (`0` = FALSE; `Non-Zero` = TRUE)
am_readwrite	INTEGER	Whether a primary access method can both read and write `0` = access method is read-only `Non-Zero` = access method is read / write
am_parallel	INTEGER	Whether an access method supports parallel execution. (`0` = FALSE; `Non-Zero` = TRUE)
am_costfactor	SMALLFLOAT	The value to be multiplied by the cost of a scan in order to normalize it to costing done for built-in access methods. The scan cost is the output of the **am_scancost** function
am_create	INTEGER	The routine specified for the AM_CREATE purpose for this access method; value = **procid** for the routine in the **sysprocedures** table
am_drop	INTEGER	The routine specified for the AM_DROP purpose function for this access method
am_open	INTEGER	The routine specified for the AM_OPEN purpose function for this access method

(2 of 3)

Column	Type	Explanation
am_close	INTEGER	The routine specified for the AM_CLOSE purpose function for this access method
am_insert	INTEGER	The routine specified for the AM_INSERT purpose function for this access method
am_delete	INTEGER	The routine specified for the AM_DELETE purpose function for this access method
am_update	INTEGER	The routine specified for the AM_UPDATE purpose function for this access method
am_stats	INTEGER	The routine specified for the AM_STATS purpose function for this access method
am_scancost	INTEGER	The routine specified for the AM_SCANCOST purpose function for this access method
am_check	INTEGER	The routine specified for the AM_CHECK purpose function for this access method
am_beginscan	INTEGER	Routine specified for the AM_BEGINSCAN purpose function for this access method
am_endscan	INTEGER	The routine specified for the AM_ENDSCAN purpose function for this access method
am_rescan	INTEGER	The routine specified for the AM_RESCAN purpose function for this access method
am_getnext	INTEGER	The routine specified for the AM_GETNEXT purpose function for this access method
am_getbyid	INTEGER	The routine specified for the AM_GETBYID purpose function for this access method
am_build	INTEGER	The routine specified for the AM_BUILD purpose function for this access method
am_init	INTEGER	The routine specified for the AM_INIT purpose function for this access method

(3 of 3)

For each of the last 18 columns (**am_create** through **am_truncate**), the value is the **sysprocedures.procid** value for the routine.

The **am_sptype** column can have multiple entries. For example:

- A means the access method supports extspaces and sbspaces. If the access method is built-in, such as a B-tree, it also supports dbspaces.
- DS means the access method supports dbspaces and sbspaces.
- sx means the access method supports sbspaces and extspaces.

A composite index on the **am_name** and **am_owner** columns in this table allows only unique values. The **am_id** column has a unique index.

For information about access method functions, refer to the documentation of your access method.

IDS

SYSATTRTYPES

The **sysattrtypes** system catalog table contains information about members of a complex data type. Each row of **sysattrtypes** contains information about elements of a collection data type or fields of a ROW data type.

The **sysattrtypes** table has the following columns.

Column	Type	Explanation
extended_id	INTEGER	Identifying code of an extended data type. Value is the same as in the **sysxtdtypes** table (page 1-79).
seqno	SMALLINT	Identifying code of an entry having **extended_id** type
levelno	SMALLINT	Position of member in collection hierarchy
parent_no	SMALLINT	Value in the **seqno** column of the complex data type that contains this member
fieldname	VARCHAR(128)	Name of the field in a ROW type. NULL for other complex data types
fieldno	SMALLINT	Field number sequentially assigned by system (from left to right within each row type)

(1 of 2)

Column	Type	Explanation
type	SMALLINT	Code for the data type; see the description of **syscolumns**.**coltype** (page 1-29).
length	SMALLINT	Length (in bytes) of the member
xtd_type_id	INTEGER	Code identifying this data type; see the description of **sysxtdtypes.extended_id** (page 1-79)

(2 of 2)

Two indexes on the **extended_id** column and the **xtd_type_id** column allow duplicate values. A composite index on the **extended_id** and **seqno** columns allows only unique values.

IDS

SYSBLOBS

The **sysblobs** system catalog table specifies the storage location of BYTE and TEXT column values. Its name is based on a legacy term for BYTE and TEXT columns (also known as *simple large objects*). The **sysblobs** table contains one row for each BYTE or TEXT column, and has the following columns.

Column	Type		Explanation
spacename	VARCHAR(128) CHAR(18)	IDS XPS	Name of partition, dbspace, or family
type	CHAR(1)		Code identifying the type of storage media: M = Magnetic O = Optical (IDS only)
tabid	INTEGER		Code identifying the table
colno	SMALLINT		Column number within its table

A composite index on **tabid** and **colno** allows only unique values.

For information about the location and size of chunks of blobspaces, dbspaces, and sbspaces for TEXT, BYTE, BLOB, and CLOB columns, see the *Administrator's Guide* and the *Administrator's Reference.*

IDS

SYSCASTS

The **syscasts** system catalog table describes the casts in the database. It contains one row for each built-in cast, each implicit cast, and each explicit cast that a user defines. The **syscasts** table has the following columns.

Column	Type	Explanation
owner	CHAR(32)	Owner of cast (user **informix** for built-in casts and *user* name for implicit and explicit casts)
argument_type	SMALLINT	Source data type on which the cast operates
argument_xid	INTEGER	Code for the source data type specified in the **argument_type** column
result_type	SMALLINT	Code for the data type returned by the cast
result_xid	INTEGER	Data type code of the data type named in the **result_type** column
routine_name	VARCHAR(128)	Function or procedure implementing the cast
routine_owner	CHAR(32)	Name of owner of the function or procedure specified in the **routine_name** column
class	CHAR(1)	Type of cast: E = Explicit cast I = Implicit cast S = Built-in cast

If **routine_name** and **routine_owner** have NULL values, it indicates that the cast is defined without a routine. This can occur if the data types specified in the **argument_type** and **result_type** columns both have the same length and alignment, and are both passed by reference or are both passed by value.

A composite index on columns **argument_type**, **argument_xid**, **result_type**, and **result_xid** allows only unique values. A composite index on columns **result_type** and **result_xid** allows duplicate values.

SYSCHECKS

The **syschecks** system catalog table describes each check constraint defined in the database. Because the **syschecks** table stores both the ASCII text and a binary encoded form of the check constraint, it contains multiple rows for each check constraint. The **syschecks** table has the following columns.

Column	Type	Explanation
constrid	INTEGER	Unique code identifying the constraint
type	CHAR(1)	Form in which the check constraint is stored: B = Binary encoded s = Select T = Text
seqno	SMALLINT	Line number of the check constraint
checktext	CHAR(32)	Text of the check constraint

The text in the **checktext** column associated with `B` type in the type column is in computer-readable format. To view the text associated with a particular check constraint, use the following query with the appropriate **constrid** code:

```
SELECT * FROM syschecks WHERE constrid=10 AND type='T'
```

Each check constraint described in the **syschecks** table also has its own row in the **sysconstraints** table.

A composite index on the **constrid**, **type**, and **seqno** columns allows only unique values.

IDS

SYSCHECKUDRDEP

The **syscheckudrdep** system catalog table describes each check constraint that is referenced by a user-defined routine (UDR) in the database. The **syscheckudrdep** table has the following columns.

Column	Type	Explanation
udr_id	INTEGER	Unique code identifying the UDR
constraint_id	INTEGER	Unique code identifying the check constraint

Each check constraint described in the **syscheckudrdep** table also has its own row in the **sysconstraints** system catalog table, where the **constrid** column has the same value as the **constraint_id** column of **syscheckudrdep**.

A composite index on the **udr_id** and **constraint_id** columns requires that combinations of these values be unique.

IDS

SYSCOLATTRIBS

The **syscolattribs** system catalog table describes the characteristics of smart large objects, namely CLOB and BLOB data types. It contains one row for each sbspace listed in the PUT clause of the CREATE TABLE statement.

The **syscolattribs** table has the following columns.

Column	Type	Explanation	
tabid	INTEGER	Code uniquely identifying the table	
colno	SMALLINT	Number of the column that contains the smart large object	
extentsize	INTEGER	Pages in smart-large-object extent, expressed in kilobytes	
flags	INTEGER	Integer representation of the combination (by addition) of hexadecimal values of the following parameters:	
		LO_NOLOG (0x00000001 = 1)	The smart large object is not logged.
		LO_LOG (0x00000010 = 2)	Logging of smart- large-objects conforms to current log mode of the database.
		LO_KEEP_LASTACCESS_TIME (0x00000100 = 4)	A record is kept of the most recent access of this smart-large-object column by a user.
		LO_NOKEEP_LASTACCESS_TIME (0x00001000 = 8)	No record is kept of the most recent access of this smart-large-object column by a user.
		HI_INTEG (0x00010000= 16)	Data pages have headers and footers to detect incomplete writes and data corruption.
		MODERATE_INTEG (Not available at this time)	Data pages do not have headers and footers.
flags1	INTEGER	Reserved for future use	
sbspace	VARCHAR(128)	Name of the sbspace	

A composite index on the **tabid**, **colno**, and **sbspace** columns allows only unique combinations of these values.

SYSCOLAUTH

The **syscolauth** system catalog table describes each set of privileges granted on a column. It contains one row for each set of column privileges granted in the database. The **syscolauth** table has the following columns.

Column	Type		Explanation
grantor	CHAR(32) CHAR(8)	IDS XPS	Name of the grantor of privilege
grantee	CHAR(32) CHAR(8)	IDS XPS	Name of the grantee of privilege
tabid	INTEGER		Code uniquely identifying the table
colno	SMALLINT		Column number within its table
colauth	CHAR(3)		3-byte pattern that specifies column privileges: s *or* S = Select u *or* U = Update r *or* R = References

If the **colauth** privilege code is uppercase (for example, S for Select), a user who has this privilege can also grant it to others. If the **colauth** privilege code is lowercase (for example, s for Select), the user who has this privilege cannot grant it to others. A hyphen (-) indicates the absence of the privilege corresponding to that position within the **colauth** pattern.

A composite index on the **tabid**, **grantor**, **grantee**, and **colno** columns allows only unique values. A composite index on the **tabid** and **grantee** columns allows duplicate values.

SYSCOLDEPEND

The **syscoldepend** system catalog table tracks the table columns specified in check and NOT NULL constraints. Because a check constraint can involve more than one column in a table, the **syscoldepend** table can contain multiple rows for each check constraint; one row is created for each column involved in the constraint. The **syscoldepend** table has the following columns.

Column	Type	Explanation
constrid	INTEGER	Code uniquely identifying the constraint
tabid	INTEGER	Code uniquely identifying the table
colno	SMALLINT	Column number within the table

A composite index on the **constrid**, **tabid**, and **colno** columns allows only unique values. A composite index on the **tabid** and **colno** columns allows duplicate values.

See also the **syscheckudrdep** system catalog table in "SYSCHECKUDRDEP" on page 1-25, which lists every check constraint that is referenced by a user-defined routine.

See also the **sysnewdepend** table in "SYSNEWDEPEND" on page 1-51, which describes the column dependencies of generalized-key indexes.

See also the **sysreferences** table in "SYSREFERENCES" on page 1-63, which describes dependencies of referential constraints.

SYSCOLUMNS

The **syscolumns** system catalog table describes each column in the database. One row exists for each column that is defined in a table or view.

Column	Type		Explanation
colname	VARCHAR(128) CHAR(18)	IDS XPS	Column name
tabid	INTEGER		Identifying code of table containing the column
colno	SMALLINT		Column number. The system sequentially assigns this (from left to right within each table)
coltype	SMALLINT		Code indicating the data type of the column: 0 = CHAR 1 = SMALLINT 2 = INTEGE R 3 = FLOAT 4 = SMALLFLOAT 5 = DECIMAL 6 = SERIAL * 7 = DATE 8 = MONEY 9 = NULL 10 = DATETIME 11 = BYTE 12 = TEXT 13 = VARCHAR 14 = INTERVAL 15 = NCHAR 16 = NVARCHAR 17 = INT8 18 = SERIAL8 * 19 = SET 20 = MULTISET 21 = LIST 22 = Unnamed ROW 40 = Variable-length opaque type 4118 = Named ROW
collength	SMALLINT		Column length (in bytes)
colmin	INTEGER		Minimum column length (in bytes)
colmax	INTEGER		Maximum column length (in bytes)
extended_id (IDS only)	INTEGER		Data type code, from the **sysxtdtypes** table, of the data type specified in the **coltype** column

* In DB-Access, an offset value of 256 is always added to these coltype codes because DB-Access sets SERIAL and SERIAL8 columns to NOT null.

XPS

Extended Parallel Server does not support the following data types: INT8, SERIAL8, SET, MULTISET, LIST, unnamed and named ROW. ♦

A composite index on **tabid** and **colno** allows only unique values.

The **coltype** codes listed on the previous page can be incremented by bitmaps showing the following features of the column.

Bit Value	Significance When Bit Is Set
0x0100	NULL values are not allowed
0x0200	Value is from a host variable
0x0400	FLOAT-to-DECIMAL for networked database server
0x0800	DISTINCT data type
0x1000	Named ROW type
0x2000	DISTINCT type from LVARCHAR base type
0x4000	DISTINCT type from BOOLEAN base type
0x8000	Collection is processed on client system

For example, the **coltype** value 4118 for named ROW types is the decimal representation of the hexadecimal value 0x1016, which is the same as the hexadecimal **coltype** value for an unnamed ROW type (0 x 016), with the named-ROW-type bit set. (The file **$INFORMIXDIR/incl/esql/sqltypes.h** contains additional information about **syscolumns.coltype** codes.)

NOT NULL Constraints

Similarly, the **coltype** value is incremented by 256 if the column does not allow NULL values. To determine the data type for such columns, subtract 256 from the value and evaluate the remainder, based on the possible **coltype** values. For example, if the **coltype** value is 262, subtracting 256 from 262 leaves a remainder of 6, indicating that the column has a SERIAL data type.

Storing Column Data Type

The database server stores the **coltype** value as bitmap, as listed in "SYSCOLUMNS" on page 1-29. This section provides additional information on **coltype** codes.

IDS

The BOOLEAN, BLOB, CLOB, and LVARCHAR data types are implemented by the database server as *built-in opaque* data types.

A built-in opaque data type is one for which the database server provides the type definition. Because these data types are built-in opaque types, they do not have a unique **coltype** value. Instead, they have one of the **coltype** values for opaque types: `41` (fixed-length opaque type), or `40` (varying-length opaque type). The different fixed-length opaque types are distinguished by the **extended_id** column in the **sysxtdtypes** system catalog table.

The following are the **coltype** values for the built-in opaque data types.

Predefined Data Type	Value for coltype Column
BLOB	41
CLOB	41
BOOLEAN	41
LVARCHAR	40

♦

Storing Column Length

The **collength** column value depends on the data type of the column.

Length of Integer-Based Columns

A **collength** value for a DATE, INTEGER, INT8, SERIAL, SERIAL8, or SMALLINT column is *not* machine-dependent. The database server uses the following lengths for these integer-based data types of SQL.

Integer-Based Data Type	Length (in bytes)
SMALLINT	2
DATE	4
INTEGER	4
SERIAL	4
INT8	8
SERIAL8	8

Length of Fixed-Point Columns

The **collength** value for a MONEY or DECIMAL (*p, s*) column can be calculated using the following formula:

```
(precision * 256) + scale
```

Length of Varying-Length Character Columns

For columns of type VARCHAR, the *max_size* and *min_space* values are encoded in the **collength** column using one of the following formulas:

- If the **collength** value is positive:

  ```
  collength = (min_space * 256) + max_size
  ```

- If the **collength** value is negative:

  ```
  collength + 65536 = (min_space * 256) + max_size
  ```

The database server uses the same formulas to encode the **collength** column for NVARCHAR data types.

Length for Time Data Types

For columns of type DATETIME or INTERVAL, **collength** is determined using the following formula:

```
(length * 256) + (first_qualifier * 16) + last_qualifier
```

The length is the physical length of the DATETIME or INTERVAL field, and *first_qualifier* and *last_qualifier* have values that the following table shows.

Field Qualifier	Value	Field Qualifier	Value
YEAR	0	FRACTION(1)	11
MONTH	2	FRACTION(2)	12
DAY	4	FRACTION(3)	13
HOUR	6	FRACTION(4)	14
MINUTE	8	FRACTION(5)	15
SECOND	10		

For example, if a DATETIME YEAR TO MINUTE column has a length of 12 (such as *YYYY:DD:MO:HH:MI)*, a *first_qualifier* value of 0 (for YEAR), and a *last_qualifier* value of 8 (for MINUTE), then the **collength** value is 3080 (from (256 * 12) + (0 * 16) + 8).

Length of Simple-Large-Object Columns

If the data type of the column is BYTE or TEXT, **collength** holds the length of the descriptor.

Storing Maximum and Minimum Values

The **colmin** and **colmax** column values hold the second-smallest and second-largest data values in the column, respectively. For example, if the values in an indexed column are 1, 2, 3, 4, and 5, the **colmin** value is 2 and the **colmax** value is 4. Storing the second-smallest and second-largest data values lets the database server make assumptions about the range of values in a given column and, in turn, further optimize searching strategies.

The **colmin** and **colmax** columns contain values only if the column is indexed and you have run the UPDATE STATISTICS statement. If you store BYTE or TEXT data in the tblspace, the **colmin** value is `-1`.

The **colmin** and **colmax** columns are valid only for data types that fit into four bytes: SMALLFLOAT, SMALLINT, INTEGER, and the first four bytes of CHAR. The values for all other noninteger column types are the initial 4 bytes of the maximum or minimum value, which are treated as an integer.

It is better to use UPDATE STATISTICS MEDIUM than to depend on **colmin** and **colmax** values. UPDATE STATISTICS MEDIUM gives better information and is valid for all data types.

IDS

The database server does not calculate **colmin** and **colmax** values for user-defined data types. These columns, however, have values for user-defined data types if a user-defined secondary access method supplies them. ♦

SYSCONSTRAINTS

The **sysconstraints** system catalog table lists the constraints placed on the columns in each database table. An entry is also placed in the **sysindexes** system catalog table (or **sysindices** view for Dynamic Server) for each unique, primary key, NOT NULL, or referential constraint that does not already have a corresponding entry in **sysindexes** or **sysindices** table. Because indexes can be shared, more than one constraint can be associated with an index. The **sysconstraints** table has the following columns.

Column	Type		Explanation
constrid	SERIAL		Code uniquely identifying the constraint
constrname	VARCHAR(128) CHAR(18)	IDS XPS	Name of the constraint
owner	CHAR(32) CHAR(8)		Name of the owner of the constraint
tabid	INTEGER		Code uniquely identifying the table

(1 of 2)

Column	Type		Explanation
constrtype	CHAR(1)		Code identifying the constraint type: C = Check constraint N = Not NULL P = Primary key R = Referential T = Table U = Unique
idxname	VARCHAR(128) CHAR(18)	IDS XPS	Name of index corresponding to the constraint

(2 of 2)

A composite index on the **constrname** and **owner** columns allows only unique values. An index on the **tabid** column allows duplicate values, and an index on the **constrid** column allows only unique values.

For check constraints (where constrtype = `C`), the idxname is always NULL. Additional information about each check constraint is contained in the syschecks and **syscoldepend** system catalog tables.

SYSDEFAULTS

The **sysdefaults** system catalog table lists the user-defined defaults that are placed on each column in the database. One row exists for each user-defined default value. The **sysdefaults** table has the following columns.

Column	Type	Explanation
tabid	INTEGER	Code uniquely identifying a table
colno	SMALLINT	Code uniquely identifying a column

(1 of 2)

Column	Type	Explanation
type	CHAR(1)	Code identifying the type of default value: C = Current L = Literal value N = NULL S = Dbservername T = Today U = User
default	CHAR(256)	If **sysdefaults.type** = L, the literal default value
class (IDS only)	CHAR(1)	Code identifying what kind of column: T = table t = row type

(2 of 2)

If a default is not explicitly specified in the CREATE TABLE statement, no entry exists in this table.

If you specify a literal for the default value, it is stored in the **default** column as ASCII text. If the literal value is not of type NCHAR, the **default** column consists of two parts. The first part is the 6-bit representation of the binary value of the default value structure. The second part is the default value in ASCII text. A blank space separates the two parts.

If the data type of the column is not NCHAR or NVARCHAR, a binary representation is encoded in the **default** column.

A composite index on the **tabid**, **colno**, and **class** columns allows only unique values. (For Extended Parallel Server, this index does not include the **class** column.)

SYSDEPEND

The **sysdepend** system catalog table describes how each view or table depends on other views or tables. One row exists in this table for each dependency, so a view based on three tables has three rows. The **sysdepend** table has the following columns.

Column	Type	Explanation
btabid	INTEGER	Code uniquely identifying the base table or view
btype	CHAR(1)	Base object type: T = Table V = View
dtabid	INTEGER	Code uniquely identifying a dependent table or view
dtype	CHAR(1)	Code for the type of dependent object; currently, only view (V = View) is implemented

The **btabid** and **dtabid** columns are indexed and allow duplicate values.

SYSDISTRIB

The sysdistrib system catalog table stores data-distribution information for the database server to use. Data distributions provide detailed table-column information to the optimizer to improve the choice of execution paths of SELECT statements. The **sysdistrib** table has the following columns.

Column	Type	Explanation
tabid	INTEGER	Code uniquely identifying the table where data values were gathered
colno	SMALLINT	Column number in the source table
seqno	INTEGER	Sequence number for multiple entries
constructed	DATE	Date when the data distribution was created

(1 of 2)

Column	Type		Explanation
mode	CHAR(1)		Optimization level: M = Medium H = High
resolution	SMALL-FLOAT		Specified in the UPDATE STATISTICS statement
confidence	SMALL-FLOAT		Specified in the UPDATE STATISTICS statement
encdat	STAT CHAR(256)	IDS XPS	Statistics information
type (IDS only)	CHAR(1)		Type of statistics: A = **encdat** has ASCII-encoded histogram in fixed-length character field S = **encdat** has user-defined statistics

(2 of 2)

Information is stored in the sysdistrib table when an UPDATE STATISTICS statement with mode MEDIUM or HIGH is executed for a table. (UPDATE STATISTICS LOW does not insert a value into the **mode** column.)

Only user **informix** can select the **encdat** column.

Each row in the **sysdistrib** system catalog table is keyed by the **tabid** and **colno** for which the statistics are collected.

For built-in data type columns, the **type** field is set to A. The **encdat** column stores an ASCII-encoded histogram that is broken down into multiple rows, each of which contains 256 bytes.

IDS

For columns of user-defined data types, the **type** field is set to S. The **encdat** column stores the statistics collected by the **statcollect** user-defined routine in multirepresentational form. Only one row is stored for each **tabid** and **colno** pair.

A composite index on the **tabid**, **colno**, and **seqno** columns requires unique combinations of values. ♦

IDS

SYSDOMAINS

The **sysdomains** table is a view on other system catalog tables. The **sysdomains** table has the following columns.

Column	Type	Explanation
id	SERIAL	Unique code identifying the domain
owner	CHAR(32)	Name of the owner of the domain
name	VARCHAR(128)	Name of the domain
type	SMALLINT	Code identifying the type of domain

There is no index on this table.

IDS

SYSERRORS

The **syserrors** system catalog table stores information about error, warning, and informational messages returned by DataBlade modules and user-defined routines using the **mi_db_error_raise( )** DataBlade API function.

The **syserrors** table has the following columns.

Column	Type	Explanation
sqlstate	CHAR(5)	SQLSTATE value associated with the error For more information about SQLSTATE values and their meanings, see the GET DIAGNOSTICS statement in the *Informix Guide to SQL: Syntax*.
locale	CHAR(36)	The locale with which this version of the message is associated (for example, '**en_us.8859-1**')
level	SMALLINT	Reserved for future use
seqno	SMALLINT	Reserved for future use
message	VARCHAR(255)	Message text

To create a new message, insert a row directly into the **syserrors** table. By default, all users can view this table, but only users with the DBA privilege can modify it.

A composite index on the **sqlstate**, **locale**, **level,** and **seqno** columns allows only unique values.

XPS

SYSEXTCOLS

The **sysextcols** system catalog table contains a row that describes each of the internal columns in external table **tabid** of format type (**fmttype**) FIXED. The **sysextcols** table has the following columns.

Column	Type	Explanation
tabid	INTEGER	Unique identifying code of a table
colno	SMALLINT	Code identifying the column
exttype	SMALLINT	Code identifying an external column type
extstart	SMALLINT	Starting position of column in the external data file
extlength	SMALLINT	External column length (in bytes)
nullstr	CHAR(256)	Represents NULL in external data
picture	CHAR(256)	Reserved for future use
decimal	SMALLINT	Precision for external decimals
extstype	CHAR(18)	External type name

No entries are stored in **sysextcols** for DELIMITED or Informix-format external files.

You can use the DBSCHEMA utility to write out the description of the external tables. To query these system catalog tables about an external table, use the **tabid** as stored in **systables** with **tabtype** = 'E'.

An index on the **tabid** column allows duplicate values.

XPS

SYSEXTDFILES

For each external table, at least one row exists in the **sysextdfiles** system catalog table, which has the following columns.

Column	Type	Explanation
tabid	INTEGER	Unique identifying code of an external table
dfentry	CHAR(152)	Data file entry

You can use DBSCHEMA to write out the description of the external tables. To query these system catalog tables about an external table, use the **tabid** as stored in **systables** with **tabtype** = 'E'.

An index on the **tabid** column allows duplicate values.

XPS

SYSEXTERNAL

For each external table, a single row exists in the **sysexternal** system catalog table. The **tabid** column associates the external table in this system catalog table with an entry in **systables**.

Column	Type	Explanation
tabid	INTEGER	Unique identifying code of an external table
fmttype	CHAR(1)	Type of format: D = (delimited) F = (fixed) I = (Informix)
codeset	CHAR(18)	ASCII, EBCDIC
recdelim	CHAR(4)	The record delimiter
flddelim	CHAR(4)	The field delimiter
datefmt	CHAR(8)	Reserved for future use
moneyfmt	CHAR(20)	Reserved for future use

(1 of 2)

Column	Type	Explanation
maxerrors	INTEGER	Number of errors to allow per coserver
rejectfile	CHAR(128)	Name of reject file
flags	INTEGER	Optional **load** flags
ndfiles	INTEGER	Number of data files in **sysextdfiles**

(2 of 2)

You can use DBSCHEMA to write out the description of the external tables. To query these system catalog tables about an external table, use the **tabid** as stored in **systables** with **tabtype** = 'E'.

An index on the **tabid** column allows only unique values.

IDS

SYSFRAGAUTH

The **sysfragauth** system catalog table stores information about the privileges that are granted on table fragments. This table has the following columns.

Column	Type	Explanation
grantor	CHAR(32)	Name of the grantor of privilege
grantee	CHAR(32)	Name of the grantee of privilege
tabid	INTEGER	Identifying code of the fragmented table
fragment	VARCHAR(128)	Name of dbspace where fragment is stored
fragauth	CHAR(6)	A 6-byte pattern that specifies fragment privileges (including 3 bytes reserved for future use): u *or* U = Update i *or* I = Insert d *or* D = Delete

In the **fragauth** column, an uppercase code (such as U for Update) means that the grantee can grant the privilege to other users; a lowercase (for example, u for Update) means the user cannot grant the privilege to others. Hyphen (**-**) indicates the absence of the privilege for that position within the pattern.

A composite index on the **tabid**, **grantor**, **grantee**, and **fragment** columns allows only unique values. A composite index on the **tabid** and **grantee** columns allows duplicate values.

The following example displays the fragment-level privileges for one base table, as they appear in the **sysfragauth** table. In this example, the grantee **ted** can grant the Update, Delete, and Insert privileges to other users.

grantor	grantee	tabid	fragment	fragauth
dba	dick	101	dbsp1	-ui---
dba	jane	101	dbsp3	--i---
dba	mary	101	dbsp4	--id--
dba	ted	101	dbsp2	-UID--

SYSFRAGMENTS

The **sysfragments** system catalog table stores fragmentation information for tables and indexes. One row exists for each table or index fragment.

The **sysfragments** table has the following columns.

Column	Type	Explanation
fragtype	CHAR(1)	Code indicating the type of fragmented object: I = Index i = Duplicated index fragment (XPS) T = Table t = Duplicated table fragment (XPS) B = TEXT or BYTE data (XPS) i = Index fragments of a duplicated table (XPS) d = data fragments of a duplicated table (XPS)
tabid	INTEGER	Unique identifying code of table

(1 of 3)

Column	Type		Explanation
indexname	VARCHAR(128) CHAR(18)	IDS XPS	Name of index
colno	INTEGER		Identifying code of TEXT or BYTE column (XPS) Identifying code of replica identifier (XPS)
partn	INTEGER		Identifying code of physical location
strategy	CHAR(1)		Code for type of fragment distribution strategy: R = Round-robin fragmentation strategy E = Expression-based fragmentation strategy I = IN DBSPACE clause specifies a specific location as part of fragmentation strategy T = Table-based fragmentation strategy H = Hash-based fragmentation strategy (XPS)
location	CHAR(1)		Reserved for future use; shows L for local
servername	VARCHAR(128) CHAR(18)	IDS XPS	Reserved for future use
evalpos	INTEGER		Position of fragment in the fragmentation list
exprtext	TEXT		Expression for fragmentation strategy (IDS) Contains names of the columns that are hashed and composite information for hybrid fragmentation strategies; shows hashed columns followed by the fragmentation expression of the dbslice. (XPS)
exprbin	BYTE		Binary version of expression
exprarr	BYTE		Range-partitioning data to optimize expression in range-expression fragmentation strategy
flags	INTEGER		Used internally (IDS) Bitmap indicating a hybrid fragmentation strategy (value = 0x00000010). Also, an additional flag (value = 0x00000020) is set on the first fragment of a globally detached index. (XPS)

(2 of 3)

Column	Type		Explanation
dbspace	VARCHAR(128) CHAR(18)	IDS XPS	Name of dbspace for fragment
levels	SMALLINT		Number of B+ tree index levels
npused	INTEGER		For table-fragmentation strategy, **npused** is the number of data pages; for index-fragmentation strategy, **npused** is the number of leaf pages.
nrows	INTEGER		For tables, **nrows** represents the number of rows in the fragment; for indexes, **nrows** represents the number of unique keys.
clust	INTEGER		Degree of index clustering; smaller numbers correspond to greater clustering
hybdpos	INTEGER		Contains the relative position of the hybrid fragment within a dbslice or list of dbspaces associated with a particular expression (IDS) The hybrid fragmentation strategy and the set of fragments against which the hybrid strategy is applied determines the relative position. The first fragment has a **hybdpos** value of zero (0). (XPS)

(3 of 3)

The **strategy** type `T` is used for attached indexes. (This is a fragmented index whose fragmentation is the same as the table fragmentation.)

IDS

A composite index on the **fragtype**, **tabid**, **indexname**, and **evalpos** columns allows duplicate values. ♦

XPS

A composite index on the **fragtype**, **tabid**, **indexname**, **evalpos**, and **hybdpos** columns allows duplicate values. ♦

SYSINDEXES

The **sysindexes** table is a view on the SYSINDICES table. It contains one row for each index in the database. The **sysindexes** table has the following columns.

Column	Type		Explanation
idxname	VARCHAR(128) CHAR(18)	IDS XPS	Index name
owner	CHAR(32) char(8)	IDS XPS	Name of owner of index (**informix** for system catalog tables and *user* name for database tables) (IDS only)
tabid	INTEGER		Unique identifying code of table
idxtype	CHAR(1)		Index type: U = Unique D = Duplicates allowed G = Nonbitmap generalized-key index (XPS) g = Bitmap generalized-key index (XPS) u = unique, bitmap (XPS) d = nonunique, bitmap (XPS)
clustered	CHAR(1)		Clustered or nonclustered index (C = Clustered)
part1	SMALLINT		Column number (**colno**) of a single index or the 1st component of a composite index
part2	SMALLINT		2nd component of a composite index
part3	SMALLINT		3rd component of a composite index
part4	SMALLINT		4th component of a composite index
part5	SMALLINT		5th component of a composite index
part6	SMALLINT		6th component of a composite index
part7	SMALLINT		7th component of a composite index
part8	SMALLINT		8th component of a composite index

(1 of 2)

Column	Type	Explanation
part9	SMALLINT	9th component of a composite index
part10	SMALLINT	10th component of a composite index
part11	SMALLINT	11th component of a composite index
part12	SMALLINT	12th component of a composite index
part13	SMALLINT	13th component of a composite index
part14	SMALLINT	14th component of a composite index
part15	SMALLINT	15th component of a composite index
part16	SMALLINT	16th component of a composite index
levels	SMALLINT	Number of B-tree levels
leaves	INTEGER	Number of leaves
nunique	INTEGER	Number of unique keys in the first column
clust	INTEGER	Degree of clustering: smaller numbers correspond to greater clustering
idxflags	INTEGER	Bitmap storing the current locking mode of the index: Normal = 0x00000001 (XPS only) Coarse = 0x00000002 (XPS only)

(2 of 2)

As with most system catalog tables, changes that affect existing indexes are reflected in this table only after you run the UPDATE STATISTICS statement.

Each **part1** through **part16** column in this table holds the column number (**colno**) of one of the 16 possible parts of a composite index. If the component is ordered in descending order, the colno is entered as a negative value. The columns are filled in for B-tree indexes that do not use user-defined types or functional indexes. For generic B-trees and all other access methods, the part1 through part16 columns all contain zeros.

The **clust** column is blank until the UPDATE STATISTICS statement is run on the table. The maximum value is the number of rows in the table, and the minimum value is the number of data pages in the table.

XPS

The **tabid** column is indexed and allows duplicate values. A composite index on the **idxname**, **owner**, and **tabid** columns allows only unique values. ♦

IDS

SYSINDICES

The **sysindices** system catalog table describes the indexes in the database. It contains one row for each index that is defined in the database. The **sysindices** table has the following columns.

Column	Type	Explanation
idxname	VARCHAR(128)	Name of index
owner	CHAR(32)	Name of owner of index (user **informix** for system catalog tables and *username* for database tables)
tabid	INTEGER	Unique identifying code of table
idxtype	CHAR(1)	Index type: U = Unique D = Duplicates allowed
clustered	CHAR(1)	Clustered or nonclustered index (C = Clustered)
levels	SMALLINT	Number of tree levels
leaves	INTEGER	Number of leaves
nunique	INTEGER	Number of unique keys in the first column
clust	INTEGER	Degree of clustering: smaller numbers correspond to greater clustering. The maximum value is the number of rows in the table, and the minimum value is the number of data pages in the table. This column is blank until the UPDATE STATISTICS statement is run on the table.
nrows	FLOAT	Estimated number of rows in the table (zero until UPDATE STATISTICS is run on the table).

(1 of 2)

Column	Type	Explanation
indexkeys	INDEX-KEYARRAY	This column has a maximum of three fields, displayed in the following form: <*function id*>(*col1*, ... , *coln*) [*operator class id*]
amid	INTEGER	Unique identifying code of the access method that implements this index. (Value = **am_id** for that access method in the **sysams** table.)
amparam	LVARCHAR	List of parameters used to customize the behavior of this access method.

(2 of 2)

***Tip:** This system catalog table is changed from the 7.2 version of Informix database servers. The earlier schema of this system catalog table is still available as a view and can be accessed under its original name: sysindexes.*

Changes that affect existing indexes are reflected in this system catalog table only after you run the UPDATE STATISTICS statement.

The fields within the **indexkeys** columns have the following significance:

- The *function id* appears only if the index is on return values of a function that is defined over the columns of the table. (That is, if the index is a functional index.) Here the *function id* is the same as the **procid** value for the function in the **sysprocedures** table.
- The list of the columns (*col1*, ... , *coln*) in the second field identifies the columns over which the index is defined.
- The *operator class id* shows the secondary access method that is used to build and to search the index. Here the *operator class id* is the same as the **opclassid** of the access method in the **sysopclasses** table.

The **tabid** column is indexed and allows duplicate values. A composite index on the **idxname**, **owner**, and **tabid** columns allows only unique values.

IDS

SYSINHERITS

The **sysinherits** system catalog table stores information about table and named ROW type inheritance. Every supertype, subtype, supertable, and subtable in the database has a corresponding row in the **sysinherits** table.

Column	Type	Explanation
child	INTEGER	Identifying code of the subtable or subtype
parent	INTEGER	Identifying code of the supertable or supertype
class	CHAR(1)	Inheritance class: t = named row type T = table

The **child** and **parent** values are from **sysxtdtypes.extended_id** for named ROW types, or from **systables.tabid** for tables. Simple indexes on the **child** and **parent** columns allow duplicate values.

IDS

SYSLANGAUTH

The **syslangauth** system catalog table contains the authorization information on computer languages that are used to write user-defined routines (UDRs).

Column	Type	Explanation
grantor	CHAR(32)	Name of the grantor of the language authorization
grantee	CHAR(32)	Name of the grantee of the language authorization
langid	INTEGER	Identifying code of language in **sysroutinelangs** table
langauth	CHAR(1)	The language authorization u = Usage privilege granted U = Usage privilege granted WITH GRANT OPTION

A composite index on the **langid**, **grantor**, and **grantee** columns allows only unique values. A composite index on the **langid** and **grantee** columns allows duplicate values.

IDS

SYSLOGMAP

The **syslogmap** system catalog table contains fragmentation information.

Column	Type	Explanation
tabloc	INTEGER	Code for the location of an external table
tabid	INTEGER	Unique identifying code of the table
fragid	INTEGER	Identifying code of the fragment
flags	INTEGER	Bitmap of modifiers from declaration of fragment

A simple index on the **tabloc** column and a composite index on the **tabid** and **fragid** columns do not allow duplicate values.

XPS

SYSNEWDEPEND

The **sysnewdepend** system catalog table contains information about generalized-key indexes that is not available in the **sysindexes** table. The dependencies between a generalized-key index and the tables in the FROM clause of the CREATE INDEX statement are stored in the **sysnewdepend** table, which has the following columns.

Column	Type	Explanation
scrid1	char(18)	Name of the generalized-key index
scrid2	INTEGER	Unique identifying code (= **tabid**) of the indexed table
type	INTEGER	Code for the type of generalized-key index
destid1	INTEGER	The **systables.tabid** value for the table on which the generalized-key index depends
destid2	INTEGER	The column number within the **destid1** table

A composite index on the **scrid1**, **scrid2**, and **type** columns allows duplicate values. Another composite index on the **destid1**, **destid2**, and **type** columns also allows duplicate values.

IDS

SYSOBJSTATE

The **sysobjstate** system catalog table stores information about the state (object mode) of database objects. The types of database objects that are listed in this table are indexes, triggers, and constraints.

Every index, trigger, and constraint in the database has a corresponding row in the **sysobjstate** table if a user creates the object. Indexes that the database server creates on the system catalog tables are not listed in the **sysobjstate** table because their object mode cannot be changed.

The **sysobjstate** table has the following columns.

Column	Type	Explanation
objtype	CHAR(1)	Code for the type of database object: C = Constraint I = Index T = Trigger
owner	CHAR(32)	Name of the owner of the database object
name	VARCHAR(128)	Name of the database object
tabid	INTEGER	Identifying code of table on which the object is defined
state	CHAR(1)	The current state (object mode) of the database object. This value can be one of the following codes: D = Disabled E = Enabled F = Filtering with no integrity-violation errors G = Filtering with integrity-violation errors

A composite index on the **objtype**, **name**, **owner**, and **tabid** columns allows only unique combinations of values. A simple index on the **tabid** column allows duplicate values.

IDS

SYSOPCLASSES

The **sysopclasses** system catalog table contains information about operator classes associated with secondary access methods. It contains one row for each operator class that has been defined in the database. The **sysopclasses** table has the following columns.

Column	Type	Explanation
opclassname	VARCHAR(128)	Name of the operator class
owner	CHAR(32)	Name of the owner of the operator class
amid	INTEGER	Identifying code of the secondary access method associated with this operator class
opclassid	SERIAL	Identifying code of the operator class
ops	LVARCHAR	List of names of the operators that belong to this operator class
support	LVARCHAR	List of names of support functions defined for this operator class

The **opclassid** value corresponds to the **sysams.am_defopclass** value that specifies the default operator class for the secondary access method that the **amid** column specifies.

The **sysopclasses** table has a composite index on the **opclassname** and **owner** columns and an index on **opclassid** column. Both indexes allow only unique values.

IDS

SYSOPCLSTR

The **sysopclstr** system catalog table defines each optical cluster in the database. It contains one row for each optical cluster. The **sysopclstr** table has the following columns.

Column	Type		Explanation
owner	CHAR(32) CHAR(8)	IDS XPS	Name of the owner of the optical cluster
clstrname	VARCHAR(128) CHAR(18)	IDS XPS	Name of the optical cluster
clstrsize	INTEGER		Size of the optical cluster
tabid	INTEGER		Unique identifying code for the table
blobcol1	SMALLINT		BYTE or TEXT column number 1
blobcol2	SMALLINT		BYTE or TEXT column number 2
blobcol3	SMALLINT		BYTE or TEXT column number 3
blobcol4	SMALLINT		BYTE or TEXT column number 4
blobcol5	SMALLINT		BYTE or TEXT column number 5
blobcol6	SMALLINT		BYTE or TEXT column number 6
blobcol7	SMALLINT		BYTE or TEXT column number 7
blobcol8	SMALLINT		BYTE or TEXT column number 8
blobcol9	SMALLINT		BYTE or TEXT column number 9
blobcol10	SMALLINT		BYTE or TEXT column number 10
blobcol11	SMALLINT		BYTE or TEXT column number 11
blobcol12	SMALLINT		BYTE or TEXT column number 12
blobcol13	SMALLINT		BYTE or TEXT column number 13
blobcol14	SMALLINT		BYTE or TEXT column number 14
blobcol15	SMALLINT		BYTE or TEXT column number 15

(1 of 2)

Column	Type	Explanation
blobcol16	SMALLINT	BYTE or TEXT column number 16
clstrkey1	SMALLINT	Cluster key number 1
clstrkey2	SMALLINT	Cluster key number 2
clstrkey3	SMALLINT	Cluster key number 3
clstrkey4	SMALLINT	Cluster key number 4
clstrkey5	SMALLINT	Cluster key number 5
clstrkey6	SMALLINT	Cluster key number 6
clstrkey7	SMALLINT	Cluster key number 7
clstrkey8	SMALLINT	Cluster key number 8
clstrkey9	SMALLINT	Cluster key number 9
clstrkey10	SMALLINT	Cluster key number 10
clstrkey11	SMALLINT	Cluster key number 11
clstrkey12	SMALLINT	Cluster key number 12
clstrkey13	SMALLINT	Cluster key number 13
clstrkey14	SMALLINT	Cluster key number 14
clstrkey15	SMALLINT	Cluster key number 15
clstrkey16	SMALLINT	Cluster key number 16

(2 of 2)

The contents of this table are sensitive to CREATE OPTICAL CLUSTER, ALTER OPTICAL CLUSTER, and DROP OPTICAL CLUSTER statements that have been executed on databases that support optical cluster subsystems. Changes that affect existing optical clusters are reflected in this table only after you run the UPDATE STATISTICS statement.

A composite index on the **clstrname** and **owner** columns allows only unique values. A simple index on the **tabid** column allows duplicate values.

SYSPROCAUTH

The **sysprocauth** system catalog table describes the privileges granted on a procedure or function. It contains one row for each set of privileges that are granted. The **sysprocauth** table has the following columns.

Column	Type		Explanation
grantor	CHAR(32) CHAR(8)	IDS XPS	Name of grantor of privileges to access the routine
grantee	CHAR(32) CHAR(8)	IDS XPS	Name of grantee of privileges to access the routine
procid	INTEGER		Unique identifying code of the routine
procauth	CHAR(1)		Type of privilege granted on the routine: e = Execute privilege on routine E = Execute privilege WITH GRANT OPTION

A composite index on the **procid**, **grantor**, and **grantee** columns allows only unique values. A composite index on the **procid** and **grantee** columns allows duplicate values.

SYSPROCBODY

The **sysprocbody** system catalog table describes the compiled version of each procedure or function in the database. Because the **sysprocbody** table stores the text of the routine, each routine can have multiple rows. The **sysprocbody** table has the following columns.

Column	Type	Explanation
procid	INTEGER	Unique identifying code for the routine
datakey	CHAR(1)	Type of information in the **data** column D = User document text E = Creation time information L = Literal value (that is, literal number or quoted string) P = Interpreter instruction code (p-code) R = Return value type list S = Routine symbol table T = Actual routine source
seqno	INTEGER	Line number within the routine
data	CHAR(256)	Actual text of the routine

The **data** column contains actual data, which can be one of the following types:

- Encoded return values list
- Encoded symbol table
- Literal data
- P-code for the routine
- Compiled code for the routine
- Text of the routine and its documentation

A composite index on the **procid**, **datakey**, and **seqno** columns allows only unique values.

SYSPROCEDURES

The **sysprocedures** system catalog table lists the characteristics for each function and procedure in the database. It contains one row for each routine.

Each function in **sysprocedures** has a unique value, **procid**, called a *routine identifier*. Throughout the system catalog, a functions is identified by its routine identifier, not by its name.

XPS

For Extended Parallel Server, **sysprocedures** has the following columns.

Column	Type	Explanation
procname	CHAR(18)	Name of routine
owner	CHAR(8)	Name of owner (IDS only)
procid	SERIAL	Unique identifying code for the routine
mode	CHAR(1)	Mode type: D *or* d = DBA O *or* o = Owner P *or* p = Protected R *or* r = Restricted
retsize	INTEGER	Compiled size (in bytes) of values
symsize	INTEGER	Compiled size (in bytes) of symbol table
datasize	INTEGER	Compiled size (in bytes) of constant data
codesize	INTEGER	Compiled size (in bytes) of routine instruction code
numargs	INTEGER	Number of arguments to routine

A composite index on **procname** and **owner** requires unique values. ♦

IDS

For Dynamic Server, **sysprocedures** has the following columns.

Column	Type	Explanation
procname	VARCHAR(128)	Name of routine
owner	CHAR(32)	Name of owner
procid	SERIAL	Unique identifying code for the routine
mode	CHAR(1)	Mode type: D *or* d = DBA O *or* o = Owner P *or* p = Protected R *or* r = Restricted
retsize	INTEGER	Compiled size (in bytes) of returned values
symsize	INTEGER	Compiled size (in bytes) of symbol table
datasize	INTEGER	Compiled size (in bytes) of constant data
codesize	INTEGER	Compiled size (in bytes) of routine instruction code
numargs	INTEGER	Number of arguments to routine
isproc	CHAR(1)	Whether routine is a procedure or a function `t` = procedure `f` = function
specificname	VARCHAR(128)	Specific name for the routine
externalname	VARCHAR(255)	Location of the external routine. This item is language-specific in content and format.
paramstyle	CHAR(1)	Parameter style `I` = Informix
langid	INTEGER	Language code (in **sysroutinelangs** table)
paramtypes	RTNPARAM-TYPES	Data types of parameters

(1 of 2)

Column	Type	Explanation
variant	BOOLEAN	Whether the routine is VARIANT or not `t` = is VARIANT `f` = is not VARIANT
client	BOOLEAN	Reserved for future use
handlesnulls	BOOLEAN	NULL handling indicator: `t` = handles NULLs `f` =does not handle NULLs
iterator	BOOLEAN	Whether the routine is an iterator function `t` = is an iterator function `f` = is not an iterator function
percallcost	INTEGER	Amount of CPU per call; integer cost to execute UDR: cost/call - 0 -(2^31-1)
commutator	VARCHAR(128)	Name of commutator function
negator	VARCHAR(128)	Name of negator function
selfunc	VARCHAR(128)	Name of function to estimate selectivity of UDR
internal	BOOLEAN	Whether the routine can be called from SQL `t` = routine is internal, not callable from SQL `f` = routine is external, callable from SQL
class	CHAR(18)	CPU class in which routine should be executed
stack	INTEGER	Stack size in bytes required per invocation
parallel-izable	BOOLEAN	Parallelization indicator for UDR: `t` = parallelizable `f` = not parallelizable
costfunc	VARCHAR(128)	Name of cost function for UDR
selconst	SMALLFLOAT	Selectivity constant for UDR

(2 of 2)

In the **mode** column, the R mode is a special case of the O mode. A routine is in restricted (R) mode if it was created with a specified owner that is different from the routine creator. If routine statements involving a remote database are executed, the database server uses the permissions of the user that executes the routine instead of the permissions of the routine owner. In all other scenarios, R-mode routines behave the same as O-mode routines.

You cannot use the DROP FUNCTION, DROP ROUTINE, or DROP PROCEDURE statements to delete a protected routine. Protected routines are indicated by lowercase in the **mode** column. In earlier versions, protected SPL routines (which cannot be deleted) were indicated by a `p`. Starting with Version 9.0, protected SPL routines are treated as DBA routines and cannot be Owner routines. Thus `D` and `O` indicate DBA and Owner routines, and `d` and `o` indicate protected DBA and protected Owner routines.

A database server can create protected stored routines for internal use. These protected stored procedures have `p` in the **mode** column. You cannot modify, drop, or display protected stored procedures.

Important: *After a SET SESSION AUTHORIZATION is done, all owner routines created while using the new identity are given a restricted mode.*

A database server can create protected routines for internal use. The **sysprocedures** table identifies these protected routines with the letter `P` or `p` in the **mode** column. You cannot modify or drop protected routines, nor can you display them through DBSCHEMA.

A unique index is on the **procid** column. A composite index on the **procname**, **isproc**, **numargs**, and **owner** columns allows duplicate values, as does a composite index on the **specificname** and **owner** columns. ♦

SYSPROCPLAN

The **sysprocplan** system catalog table describes the query-execution plans and dependency lists for data-manipulation statements within each routine. Because different parts of a routine plan can be created on different dates, this table can contain multiple rows for each routine.

The **sysprocplan** table has the following columns.

Column	Type	Explanation
procid	INTEGER	Unique identifying code for the routine
planid	INTEGER	Unique identifying code for the plan
datakey	CHAR(1)	Type of information stored in **data** column D = Dependency list I = Information record Q = Execution plan
seqno	INTEGER	Line number within the plan
created	DATE	Date when plan was created
datasize	INTEGER	Size (in bytes) of the list or plan
data	CHAR(256)	Encoded (compiled) list or plan (IDS) Text of the SPL routine (XPS)

Before a routine is run, its dependency list in the **data** column is examined. If the major version number of a table that the plan accesses has changed, or if any object that the routine uses has been modified since the plan was optimized (for example, if an index has been dropped), then the plan is optimized again.

When **datakey** is I, the data column stores information about UPDATE STATISTICS and PDQPRIORITY.

It is possible to delete all the plans for a given routine by using the DELETE statement on **sysprocplan**. When the routine is subsequently executed, new plans are automatically generated and recorded in **sysprocplan**.

The UPDATE STATISTICS FOR PROCEDURE statement also updates this table.

A composite index on the **procid**, **planid**, **datakey**, and **seqno** columns allows only unique values.

SYSREFERENCES

The **sysreferences** system catalog table lists all referential constraints on columns. It contains a row for each referential constraint in the database. The **sysreferences** table has the following columns.

Column	Type	Explanation
constrid	INTEGER	Code uniquely identifying the constraint
primary	INTEGER	Identifying code of the corresponding primary key
ptabid	INTEGER	Identifying code of the table that is the primary key
updrule	CHAR(1)	Reserved for future use; displays an R
delrule	CHAR(1)	Whether constraint uses cascading delete or restrict rule: C = Cascading delete R = Restrict (default)
matchtype	CHAR(1)	Reserved for future use; displays an N
pendant	CHAR(1)	Reserved for future use; displays an N

The **constrid** column is indexed and allows only unique values. The **primary** column is indexed and allows duplicate values.

XPS

SYSREPOSITORY

The **sysrepository** system catalog table contains information about generalized-key indexes that is not available in the **sysindexes** system catalog table. The **sysrepository** table has the following columns.

Column	Type	Explanation
id1	CHAR(18)	Index from the generalized-key index
id2	INTEGER	Tabid of table with the generalized-key index
type	INTEGER	Integer classifying the type of object In this release, the only value that can appear is 1, indicating a generalized-key index type.
seqid	SERIAL	For future use. Unrelated to **syssequences.seqid**
desc	TEXT	The CREATE statement of a generalized-key index
bin	BYTE	Internal representation of the generalized-key index

The contents of the **sysrepository** table are useful when a generalized-key index has to be rebuilt during a recovery, or if a user wants to see the CREATE statement for a specific generalized-key index.

The contents of the **sysrepository** table are useful when a generalized-key index has to be rebuilt during a recovery, or if a user wants to see the CREATE statement for a specific generalized-key index.

The **desc** column contains the CREATE statement for each generalized-key index in the database.

An index on the **seqid** column allows duplicate values. A composite index on the **id1**, **id2**, and **type** columns requires unique combinations of values.

IDS

SYSROLEAUTH

The **sysroleauth** system catalog table describes the roles that are granted to users. It contains one row for each role that is granted to a user in the database. The **sysroleauth** table has the following columns.

Column	Type	Explanation
rolename	CHAR(32)	Name of the role
grantee	CHAR(32)	Name of the grantee of the role
is_grantable	CHAR(1)	Specifies whether the role is grantable: Y = Grantable N = Not grantable

The **is_grantable** column indicates whether the role was granted with the WITH GRANT OPTION of the GRANT statement.

A composite index on the **rolename** and **grantee** columns allows only unique values.

IDS

SYSROUTINELANGS

The **sysroutinelangs** system catalog table lists the supported programming - languages for writing user-defined routines (UDRs). The **sysroutinelangs** table has the following columns.

Column	Type	Explanation
langid	SERIAL	Code uniquely identifying a supported language
langname	CHAR(30)	Name of the language, such as C or SPL
langinitfunc	VARCHAR(128)	Name of initialization function for the language
langpath	CHAR(255)	Directory path for the UDR language
langclass	CHAR(18)	Name of the class of the UDR language

An index on the **langname** column allows duplicate values.

SYSSYNONYMS

The **syssynonyms** system catalog table lists the synonyms for each table, view, or sequence. The **syssynonyms** table contains a row for every synonym that is defined in the database, and has the following columns.

Column	Type		Explanation
owner	CHAR(32) CHAR(8)	IDS XPS	Name of the owner of the synonym
synname	VARCHAR(128) CHAR(18)	IDS XPS	Name of the synonym
created	DATE		Date when the synonym was created
tabid	INTEGER		Identifying code of a table, view, or sequence

A composite index on the **owner** and **synonym** columns allows only unique values. The **tabid** column is indexed and allows duplicate values.

SYSSYNTABLE

The **syssyntable** system catalog table outlines the mapping between each public synonym and the database object (a table, view, or sequence) that it represents. It contains one row for each entry in the **systables** table that has a **tabtype** value of S. The **syssyntable** table has the following columns.

Column	Type		Explanation
tabid	INTEGER		Identifying code of the public synonym
servername	VARCHAR(128) CHAR(18)	IDS XPS	Name of an external database server
dbname	VARCHAR(128) CHAR(18)	IDS XPS	Name of an external database

(1 of 2)

Column	Type		Explanation
owner	CHAR(32) CHAR(8)	IDS XPS	Name of the owner of an external object
tabname	VARCHAR(128) CHAR(18)	IDS XPS	Name of an external table, view, or sequence
btabid	INTEGER		Identifying code of a base table, view, or sequence

(2 of 2)

If you define a synonym for a table that is in your current database, only the **tabid** and **btabid** columns are used. If you define a synonym for a table that is external to your current database, the **btabid** column is not used, but the **tabid**, **servername**, **dbname**, **owner**, and **tabname** columns are used.

The **tabid** column maps to **systables.tabid**. With the **tabid** information, you can determine additional facts about the synonym from **systables**.

An index on the **tabid** column allows only unique values. The **btabid** column is indexed to allow duplicate values.

In an ANSI-compliant database, public synonyms are not supported; for this reason, the **syssyntable** table remains empty.

IDS

SYSTABAMDATA

The **systabamdata** system catalog table stores the parameter options (table-specific hashing parameters) that you chose when you created a table using a primary access method.

The **systabamdata** table has the following columns.

Column	Type	Explanation
tabid	INTEGER	Identifying code of the table
am_param	CHAR(256)	Access method parameter choices
am_space	VARCHAR(128)	Name of the space where the data values are stored

The **am_param** column stores configuration parameters that determine how a primary access method accesses a given table. Each configuration parameter in the **am_param** list has the format *keyword=value* or *keyword*.

The **am_space** column specifies the location of the table. It might reside in a cooked file, a different database, or an sbspace within the database server.

The **tabid** column is the primary key to the **systables** table. This column is indexed and must contain unique values.

SYSTABAUTH

The **systabauth** system catalog table describes each set of privileges that are granted on a table (or on a view, synonym, or sequence). It contains one row for each set of table privileges that are granted in the database; the REVOKE statement can modify a row. The **systabauth** table has the following columns.

Column	Type		Explanation
grantor	CHAR(32) CHAR(8)	IDS XPS	Name of the grantor of privilege (IDS only)
grantee	CHAR(32) CHAR(8)	IDS XPS	Name of the grantee of privilege (IDS only)
tabid	INTEGER		Identifying code of the table, view, synonym, or sequence
tabauth	CHAR(9) CHAR(8)	IDS XPS	Pattern that specifies privileges on the object: s *or* S = Select u *or* U = Update ***** = Column-level privilege; see **syscolauth** (page 1-46) i *or* I = Insert d *or* D = Delete x *or* X = Index a *or* A = Alter r *or* R = References n *or* N = Under privilege (IDS)

If the **tabauth** column shows a privilege code in uppercase (for example, S for select), this indicates that the user also has the option to grant that privilege to others. Privilege codes listed in lowercase (for example, s for select) indicate that the user has the specified privilege, but cannot grant it to others.

A hyphen (-) indicates the absence of the privilege corresponding to that position within the **tabauth** pattern.

A **tabauth** value with an asterisk (*) symbol means column-level privileges exist. (In DB-Access, the **Privileges** option of the **Info** command for a specified table can display the column-level privileges on that table.)

A composite index on **tabid**, **grantor**, and **grantee** allows only unique values. A composite index on **tabid** and **grantee** allows duplicate values.

SYSTABLES

The **systables** system catalog table contains one row for each table, view, sequence, or synonym defined in the database, including the system catalog.

Column	Type		Explanation
tabname	VARCHAR(128) CHAR(18)	IDS XPS	Name of table, view, sequence, or synonym
owner	CHAR(32) CHAR(8)	IDS XPS	Name of owner of table (**informix** for system catalog tables and *user* name for database tables)
partnum	INTEGER		Physical location code
tabid	SERIAL		System-assigned sequential identifying number
rowsize	SMALLINT		Row size
ncols	SMALLINT		Number of columns in the table
nindexes	SMALLINT		Number of indexes on the table
nrows	INTEGER		Number of rows in the table
created	DATE		Date when the table was created

(1 of 2)

Column	Type		Explanation
version	INTEGER		Number that changes when table is altered
tabtype	CHAR(1)		Code indicating the type of object: T = Table E = External Table V = View P = Private synonym S = Public synonym (Type S is not available in an ANSI-compliant database.)
locklevel	CHAR(1)		Lock mode for the table: B = Page P = Page R = Row T = Table (XPS)
npused	INTEGER		Number of data pages that have ever been initialized in the tablespace by the database server
fextsize	INTEGER		Size of initial extent (in kilobytes)
nextsize	INTEGER		Size of all subsequent extents (in kilobytes)
flags	SMALLINT		Codes for classifying permanent tables: ST_RAW (= `0x00000010`) (IDS) RAW (= `0x00000002`) (XPS) STATIC (= `0x00000004`) (XPS) OPERATIONAL (= `0x00000008`) (XPS) STANDARD (= `0x00000010`) (XPS) EXTERNAL (= `0x00000020`) (XPS)
site	VARCHAR(128) CHAR(18)	IDS XPS	Reserved for future use
dbname	VARCHAR(128) CHAR(18)	IDS XPS	Reserved for future use
type_xid (IDS only)	INTEGER		Code from **sysxtdtypes.extended_id** for typed tables, or `0` for untyped tables.
am_id (IDS only)	INTEGER		Access method code (key to **sysams** table); NULL or 0 indicates built-in storage manager

(2 of 2)

Each table, view, synonym, and sequence recorded in the **systables** table is assigned a **tabid**, which is a system-assigned SERIAL value that uniquely identifies the object. The first 99 **tabid** numbers are reserved for system catalog tables; values for user-defined objects begin with 100.

The **tabid** column is indexed and contains only unique values. A composite index on the **tabname** and **owner** columns also requires unique values.

The version column contains an encoded number that is stored in systables when a new table is created. Portions of this value are incremented when data-definition statements, such as ALTER INDEX, ALTER TABLE, DROP INDEX, and CREATE INDEX, are performed on the table.

In the **flags** column, ST_RAW represents a nonlogging permanent table in a database that supports transaction logging.

When a prepared statement that references a database table is executed, the version value is checked to make sure that nothing has changed since the statement was prepared. If the version value has changed, the prepared statement is not executed, and you must prepare the statement again.

The **npused** column does not reflect the number of pages used for BYTE or TEXT data, nor the number of pages that are freed in DELETE operations.

GLS

The **systables** table has two rows that store information about the database locale: GL_COLLATE with a **tabid** of `90` and GL_CTYPE with a **tabid** of `91`. To view these rows, enter the following SELECT statement:

```
SELECT * FROM systables WHERE tabid=90 OR tabid=91
```

♦

IDS

SYSTRACECLASSES

The **systraceclasses** system catalog table contains the names and identifiers of trace classes. The **systraceclasses** table has the following columns.

Column	Type	Explanation
name	CHAR(18)	Name of the class of trace messages
classid	SERIAL	Identifying code of the trace class

A *trace class* is a category of trace messages that you can use in the development and testing of new DataBlade modules and user-defined routines. Developers use the tracing facility by calling the appropriate DataBlade API routines within their code.

To create a new trace class, insert a row directly into the **systraceclasses** table. By default, all users can view this table, but only users with the DBA privilege can modify it.

The database cannot support tracing unless the MITRACE_OFF configuration parameter is undefined.

A unique index on the **name** column requires each trace class to have a unique name. The database server assigns to each class a unique sequential code. The index on this **classid** column also allows only unique values.

IDS

SYSTRACEMSGS

The **systracemsgs** system catalog table stores internationalized trace messages that you can use in debugging user-defined routines.

The **systracemsgs** table has the following columns.

Column	Type	Explanation
name	VARCHAR(128)	Name of the message
msgid	SERIAL	Identifying code of the message template
locale	CHAR(36)	Locale with which this version of the message is associated (for example, **en_us.8859-1**)
seqno	SMALLINT	Reserved for future use
message	VARCHAR(255)	The message text

DataBlade module developers create a trace message by inserting a row directly into the **systracemsgs** table. Once a message is created, the development team can specify it either by name or by **msgid** code, using trace statements that the DataBlade API provides.

To create a trace message, you must specify its name, locale, and text. By default, all users can view the **systracemsgs** table, but only users with the DBA privilege can modify it.

The database cannot support tracing unless the MITRACE_OFF configuration parameter is undefined.

A unique composite index is defined on the **name** and **locale** columns. Another unique index is defined on the **msgid** column.

SYSTRIGBODY

The **systrigbody** system catalog table contains the ASCII text of the trigger definition and the linearized code for the trigger. *Linearized code* is binary data and code that is represented in ASCII format.

Important: *The database server uses the linearized code that is stored in* ***systrigbody****. You must not alter the content of rows that contain linearized code.*

The **systrigbody** table has the following columns.

Column	Type	Explanation
trigid	INTEGER	Identifying code of the trigger
datakey	CHAR(1)	Code specifying the type of data: A = ASCII text for the body, triggered actions B = Linearized code for the body D = English text for the header, trigger definition H = Linearized code for the header S = Linearized code for the symbol table
seqno	INTEGER	Page number of this data segment
data	CHAR(256)	English text or linearized code

A composite index on the **trigid**, **datakey**, and **seqno** columns allows only unique values.

SYSTRIGGERS

The **systriggers** system catalog table contains information about the SQL triggers in the database. This information includes the triggering event and the correlated reference specification for the trigger. The **systriggers** table has the following columns.

Column	Type		Explanation
trigid	SERIAL		Identifying code of the trigger
trigname	VARCHAR(128) CHAR(18)	IDS XPS	Name of the trigger
owner	CHAR(32) CHAR(8)	IDS XPS	Name of the owner of the trigger
tabid	INTEGER		Identifying code of the triggering table
event	CHAR(1)		Code for the type of triggering event: D = Delete trigger I = Insert trigger S = Select trigger U = Update trigger
old	VARCHAR(128) CHAR(18)	IDS XPS	Name of value before update
new	VARCHAR(128) CHAR(18)	IDS XPS	Name of value after update
mode	CHAR(1)		Reserved for future use

A composite index on the **trigname** and **owner** columns allows only unique values. An index on the **trigid** column also requires unique values. An index on the **tabid** column allows duplicate values.

SYSUSERS

The **sysusers** system catalog table describes each set of privileges that are granted on the database. It contains one row for each user who has privileges on the database. This system catalog table has the following columns.

Column	Type		Explanation
username	CHAR(32) CHAR(8)	IDS XPS	Name of the database user or role
usertype	CHAR(1)		Code specifying database-level privileges: C = Connect (work within existing tables) D = DBA (all privileges) G = Role R = Resource (CREATE permanent tables, user-defined data types, and indexes)
priority	SMALLINT		Reserved for future use
password	CHAR(16) CHAR(8)	IDS XPS	Reserved for future use

An index on **username** allows only unique values. The **username** value can be the login name of a user or (on Dynamic Server) the name of a role.

SYSVIEWS

The **sysviews** system catalog table describes each view in the database. Because it stores the SELECT statement that created the view, **sysviews** can contain multiple rows for each view. It has the following columns.

Column	Type	Explanation
tabid	INTEGER	Identifying code of the view
seqno	SMALLINT	Line number of the SELECT statement
viewtext	CHAR(64)	Actual SELECT statement used to create the view

A composite index on **tabid** and **seqno** allows only unique values.

SYSVIOLATIONS

The **sysviolations** system catalog table stores information about the constraint violations for base tables. Every table in the database that has a violations table and a diagnostics table associated with it has a corresponding row in the **sysviolations** table, which has the following columns.

Column	Type	Explanation
targettid	INTEGER	Identifying code of the *target table* (the base table on which the violations table and the diagnostic table are defined).
viotid	INTEGER	Identifying code of the violations table
diatid	INTEGER	Identifying code of the diagnostics table
maxrows	INTEGER	Maximum number of rows that can be inserted into the diagnostics table by a single insert, update, or delete operation on a target table that has a filtering mode object defined on it (IDS) The maximum number of rows allowed in the violations table for each coserver (XPS)

The **maxrows** column also signifies the maximum number of rows that can be inserted in the diagnostics table during a single operation that enables a disabled object or that sets a disabled object to filtering mode (provided that a diagnostics table exists for the target table). If no maximum is specified for the diagnostics or violations table, then **maxrows** contains a NULL value.

XPS

Extended Parallel Server does not use the diagnostic table when a constraint violation occurs. Rather, the database server stores additional information in the violations table. The violations table contains the data that the transaction refused and an indication of the cause. ♦

The primary key of this table is the **targettid** column. An additional unique index is also defined on the **viotid** column.

IDS

Dynamic Server also has a unique index on the **diatid** column. ♦

IDS

SYSXTDDESC

The **sysxtddesc** system catalog table provides a text description of each UDT defined in the database. The **sysxtddesc** table has the following columns.

Column	Type	Explanation
extended_id	INTEGER	Code uniquely identifying the extended data types
seqno	SMALLINT	Value to order and identify one line of the description of the UDT. A new line is created only if the remaining text string is larger than 255 bytes.
description	CHAR(256)	Textual description of the extended data type

A composite index on **extended_id** and **seqno** allows duplicate values.

IDS

SYSXTDTYPEAUTH

The **sysxtdtypeauth** system catalog table identifies the privileges for each UDT (user-defined data type). The **sysxtdtypeauth** table contains one row for each set of privileges granted and has the following columns.

Column	Type	Explanation
grantor	CHAR(32)	Name of grantor of privilege
grantee	CHAR(32)	Name of grantee of privilege
type	INTEGER	Code identifying the UDT
auth	CHAR(2)	Code identifying privileges on the UDT: n *or* N = Under privilege u *or* U = Usage privilege

If the privilege code in the **auth** column is uppercase (for example, 'U' for usage), a user who has this privilege can also grant it to others. If the code is in lowercase, a user who has the privilege cannot grant it to others.

A composite index on **type**, **grantor**, and **grantee** allows only unique values. A composite index on the **type** and **grantee** columns allows duplicate values.

IDS

SYSXTDTYPES

The **sysxtdtype** system catalog table has an entry for each UDT (user-defined data type), including OPAQUE and DISTINCT data types and complex data types (named ROW type, unnamed ROW type, and COLLECTION type) that is defined in the database. The **sysxtdtypes** table has the following columns.

Column	Type	Explanation
extended_id	SERIAL	Unique identifying code for extended data type
domain	CHAR(1)	Code for the domain of the UDT
mode	CHAR(1)	Code classifying the UDT: B = Base (opaque) type C = Collection type or unnamed row type D = Distinct type R = Named row type ' ' (blank) = Built-in type
owner	CHAR(32)	Name of the owner of the UDT
name	VARCHAR(128)	Name of the UDT
type	SMALLINT	Code classifying the UDT
source	INTEGER	The **sysxtdtypes** reference (for distinct types only). Zero (0) indicates that a distinct UDT was created from a built-in data type.
maxlen	INTEGER	The maximum length for variable-length data types. Zero indicates a fixed-length UDT.
length	INTEGER	The length in bytes for fixed-length data types. Zero indicates a variable-length UDT.
byvalue	CHAR(1)	'T' = UDT is passed by value 'F' = UDT is not passed by value
cannothash	CHAR(1)	'T' = UDT is hashable by default hash function 'F' = UDT is not hashable by default function
align	SMALLINT	Alignment (= 1, 2, 4, *or* 8) for this UDT
locator	INTEGER	Locator key for unnamed ROW type

Each extended data type is characterized by a unique identifier, called an extended identifier (**extended_id**), a data type identifier (**type**), and the length and description of the data type.

For DISTINCT types created from built-in data types, the **type** column codes correspond to the value of the **syscolumns.coltype** column (indicating the source type) as listed on page 1-29, but incremented by the hexadecimal value 0x0000800. The file **$INFORMIXDIR/incl/esql/sqltypes.h** contains information about **sysxtdtypes.type** and **syscolumns.coltype** codes.

An index on the **extended_id** column allows only unique values. An index on the **locator** column allows duplicate values, as does a composite indexes on the **name** and **owner** columns. A composite index on the **type** and **source** columns also allows duplicate values.

IDS

Information Schema

The Information Schema consists of read-only views that provide information about all the tables, views, and columns on the current database server to which you have access. In addition, Information Schema views provide information about SQL dialects (such as Informix, Oracle, or Sybase) and SQL standards.

This version of the Information Schema views are X/Open CAE standards. Informix provides them so that applications developed on other database systems can obtain Informix system catalog information without accessing the Informix system catalog tables directly.

Important: *Because the X/Open CAE standards Information Schema views differ from ANSI-compliant Information Schema views, Informix recommends that you do not install the X/Open CAE Information Schema views on ANSI-compliant databases.*

The following Information Schema views are available:

- **tables**
- **columns**
- **sql_languages**
- **server_info**

Sections that follow contain information about how to generate and access Information Schema views as well as information about their structure.

Generating the Information Schema Views

The Information Schema views are generated automatically when you, as DBA, run the following DB-Access command:

```
dbaccess database-name $INFORMIXDIR/etc/xpg4_is.sql
```

The views display data from the system catalog tables. If tables, views, or routines exist with any of the same names as the Information Schema views, you must either rename those database objects or rename the views in the script before you can install the views. You can drop the views with the DROP VIEW statement on each view. To re-create the views, rerun the script.

Important: *In addition to the columns specified for each Information Schema view, individual vendors might include additional columns or change the order of the columns. Informix recommends that applications not use the forms SELECT * or SELECT table-name* to access an Information Schema view.*

Accessing the Information Schema Views

All Information Schema views have the Select privilege granted to PUBLIC WITH GRANT OPTION so that all users can query the views. Because no other privileges are granted on the Information Schema views, they cannot be updated.

You can query the Information Schema views as you would query any other table or view in the database.

Structure of the Information Schema Views

The following views are described in this section:

- **tables**
- **columns**
- **sql_languages**
- **server_info**

In order to accept long identifier names, most of the columns in the views are defined as VARCHAR data types with large maximum sizes.

The tables Information Schema View

The **tables** Information Schema view contains one row for each table to which you have access. It contains the following columns.

Column	Data Type	Explanation
table_schema	VARCHAR(128)	Name of owner of table
table_name	VARCHAR(128)	Name of table or view
table_type	VARCHAR(128)	BASE TABLE for table or VIEW for view
remarks	VARCHAR(255)	Reserved for future use

The visible rows in the **tables** view depend on your privileges. For example, if you have one or more privileges on a table (such as Insert, Delete, Select, References, Alter, Index, or Update on one or more columns), or if privileges are granted to PUBLIC, you see the row that describes that table.

The columns Information Schema View

The **columns** Information Schema view contains one row for each accessible column. It contains the following columns.

Column	Data Type	Explanation
table_schema	VARCHAR(128)	Name of owner of table
table_name	VARCHAR(128)	Name of table or view
column_name	VARCHAR(128)	Name of the column in the table or view
ordinal_position	INTEGER	Position of the column within its table. The **ordinal_position** value is a sequential number that starts at 1 for the first column. This is an Informix extension to XPG4.
data_type	VARCHAR(254)	Name of the data type of the column, such as CHARACTER or DECIMAL
char_max_length	INTEGER	Maximum length (in bytes) for character data types; NULL otherwise
numeric_precision	INTEGER	Total number of digits allowed for exact numeric data types (DECIMAL, INTEGER, MONEY, and SMALLINT), and the number of digits of mantissa precision for approximate data types (FLOAT and SMALLFLOAT), and NULL for all other data types. The value is machine dependent for FLOAT and SMALLFLOAT.
numeric_prec_radix	INTEGER	Uses one of the following values: `2` = approximate data types (FLOAT and SMALLFLOAT) `10` = exact numeric data types (DECIMAL, INTEGER, MONEY, and SMALLINT) Null for all other data types

(1 of 2)

Column	Data Type	Explanation
numeric_scale	INTEGER	Number of significant digits to the right of the decimal point for DECIMAL and MONEY data types: 0 for INTEGER and SMALLINT types Null for all other data types
datetime_precision	INTEGER	Number of digits in the fractional part of the seconds for DATE and DATETIME columns; null otherwise. This column is an Informix extension to XPG4.
is_nullable	VARCHAR(3)	Indicates whether a column allows NULLs; either YES or NO
remarks	VARCHAR(254)	Reserved for future use

(2 of 2)

The sql_languages Information Schema View

The **sql_languages** Information Schema view contains a row for each instance of conformance to standards that the current database server supports. The **sql_languages** view contains the following columns.

Column	Data Type	Explanation
source	VARCHAR(254)	Organization that defines this SQL version
source_year	VARCHAR(254)	Year the source document was approved
conformance	VARCHAR(254)	Standard to which the server conforms
integrity	VARCHAR(254)	Indicates whether this is an integrity enhancement feature; either YES or NO

(1 of 2)

Column	Data Type	Explanation
implementation	VARCHAR(254)	Identifies the SQL product of the vendor
binding_style	VARCHAR(254)	Direct, module, or other binding style
programming_lang	VARCHAR(254)	Host language for which binding style is adopted

(2 of 2)

The **sql_languages** view is completely visible to all users.

The server_info Information Schema View

The **server_info** Information Schema view describes the database server to which the application is currently connected. It contains two columns.

Column	Data Type	Explanation
server_attribute	VARCHAR(254)	An attribute of the database server
attribute_value	VARCHAR(254)	Value of the **server_attribute** as it applies to the current database server

Each row in this view provides information about one attribute. X/Open-compliant databases must provide applications with certain required information about the database server.

The **server_info** view includes the following **server_attribute** information.

server_attribute	Explanation
identifier_length	Maximum number of bytes for a user-defined identifier
row_length	Maximum number of bytes in a row

(1 of 2)

server_attribute	Explanation
userid_length	Maximum number of bytes in a user name (or "authorization identifier")

txn_isolation	Initial transaction isolation level for the database server:
	`Read Committed`
	Default isolation level for databases with no logging
	`Read Uncommitted`
	Default isolation level for databases that are not ANSI-compliant but support transaction logging
	`Serializable`
	Default isolation level for ANSI-compliant databases

collation_seq	Assumed ordering of the character set for the database server. The following values are possible:
	`ISO 8859-1`
	`EBCDIC`
	The Informix representation shows ISO 8859-1.

(2 of 2)

The **server_info** Information Schema view is completely visible to all users.

Chapter

Data Types

2

In This Chapter 2-5

Summary of Data Types 2-5

Description of Data Types 2-9
BLOB 2-9
BOOLEAN 2-10
BYTE 2-11
CHAR(n) 2-12
CHARACTER(n) 2-14
CHARACTER VARYING(m,r) 2-14
CLOB 2-14
DATE 2-16
DATETIME 2-17
DEC 2-21
DECIMAL 2-21
Distinct 2-23
DOUBLE PRECISION 2-24
FLOAT(n). 2-24
INT 2-25
INT8 2-25
INTEGER. 2-26
INTERVAL 2-26
LIST(e). 2-29
LVARCHAR 2-31
MONEY(p,s). 2-31
MULTISET(e) 2-32
NCHAR(n) 2-34
NUMERIC(p,s) 2-34

NVARCHAR(m,r) 2-34
Opaque 2-34
REAL 2-35
Row, Named 2-35
Row, Unnamed 2-37
SERIAL(n) 2-39
SERIAL8 2-40
SET(e) 2-41
SMALLFLOAT 2-43
SMALLINT 2-43
TEXT 2-44
VARCHAR(m,r) 2-46

Built-In Data Types 2-48
Large-Object Data Types. 2-48
Simple Large Objects 2-49
Smart Large Objects 2-50
Time Data Types 2-51
Manipulating DATETIME Values 2-52
Manipulating DATETIME with INTERVAL Values 2-53
Manipulating DATE with DATETIME and INTERVAL Values . 2-54
Manipulating INTERVAL Values 2-56
Multiplying or Dividing INTERVAL Values. 2-57

Extended Data Types 2-57
Complex Data Types 2-58
Collection Data Types 2-59
Row Data Types 2-60
DISTINCT Data Types 2-61
OPAQUE Data Types 2-61

Data Type Casting and Conversion 2-62
Using Built-in Casts 2-63
Converting from Number to Number 2-64
Converting Between Number and CHAR 2-64
Converting Between INTEGER and DATE or DATETIME . . . 2-65
Converting Between DATE and DATETIME 2-65
Using User-Defined Casts 2-66
Implicit Casts 2-66
Explicit Casts 2-66

Determining Which Cast to Apply 2-67
Casts for Distinct Types 2-68
What Extended Data Types Can Be Cast? 2-69

Operator Precedence 2-70

In This Chapter

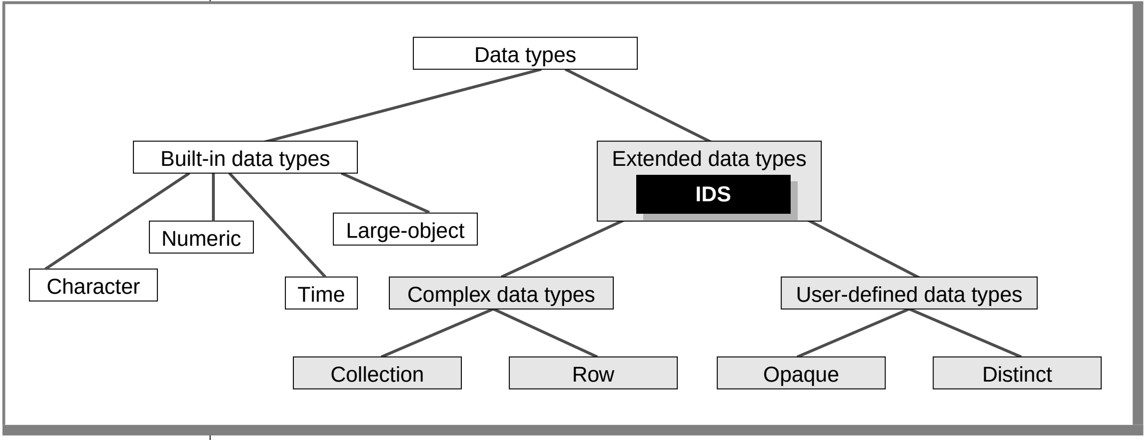
Every column in a table in a database is assigned a data type. The data type precisely defines the kinds of values that you can store in that column.

This chapter built-in and extended data types, casting between two data types, and operator precedence.

Summary of Data Types

Figure 2-1 shows the logical categories of data types that Informix database servers support. The shaded categories are for Dynamic Server only.

Figure 2-1
Overview of Supported Data Types



Built-in data types (which are system-defined) and *extended* data types (which you can define) share the following characteristics. You can:

- Use them to create columns within database tables.
- Declare them as arguments and as return types of routines.
- Use them as base types from which to create DISTINCT types.
- Cast them to other data types.
- Declare and access host variables of these types in SPL and ESQL/C.

Specific exceptions are mentioned in the description of each data type. For an overview, see "Built-In Data Types" on page 2-48 and "Extended Data Types" on page 2-57.

You assign data types to columns with the CREATE TABLE statement and change them with the ALTER TABLE statement. When you change an existing column data type, all data is converted to the new data type, if possible.

For information on the ALTER TABLE and CREATE TABLE statements, SQL statements that create specific data types and create and drop casts, and other data type syntax conventions, refer to the *Informix Guide to SQL: Syntax*.

IDS

For information about how to create and use complex data types, see the *Informix Guide to Database Design and Implementation*. For information about how to create user-defined data types, see *Creating User-Defined Routines and User-Defined Data Types*. ♦

All Informix database servers support the data types that Figure 2-2 lists. This chapter describes each of these data types.

Figure 2-2
Data Types That All Informix Database Servers Support

Data Type	Explanation	Page
BYTE	Stores any kind of binary data, up to 2^{31} bytes in length	2-11
CHAR(*n*)	Stores character strings; collation is in code-set order	2-12
CHARACTER(*n*)	Is a synonym for CHAR	2-14
CHARACTER VARYING(*m*,*r*)	Stores character strings of varying length (ANSI compliant); collation is in code-set order	2-14

(1 of 3)

Data Type	Explanation	Page
DATE	Stores calendar date	2-16
DATETIME	Stores calendar date combined with time of day	2-17
DEC	Is a synonym for DECIMAL	2-20
DECIMAL(*p*)	Stores floating-point numbers with definable precision; if database is ANSI-compliant, the scale is zero.	2-21
DECIMAL(*p, s*)	Stores fixed-point numbers with definable scale and precision	2-21
DOUBLE PRECISION	Synonym for FLOAT	2-24
FLOAT(*n*)	Stores double-precision floating-point numbers corresponding to the **double** data type in C	2-24
INT	Is a synonym for INTEGER	2-25
INTEGER	Stores whole numbers in a range from –2,147,483,647 to +2,147,483,647	2-25
INTERVAL (*Year* | *Month*)	Stores a span of time (or level of effort) in units of *years* and / or *months*.	2-26
INTERVAL (*Day* | *Fraction*)	Stores a span of time in a contiguous set of units of *days*, *hours*, *minutes*, *seconds*, and / or *fractions of a second*.	2-26
MONEY(*p,s*)	Stores currency amount	2-31
NCHAR(*n*)	Stores character strings; collation is locale dependent	2-34
NUMERIC(*p,s*)	Synonym for DECIMAL(*p,s*)	2-34
NVARCHAR(*m,r*)	Stores character strings of varying length (up to 255 bytes); collation is locale dependent	2-34
REAL	Is a synonym for SMALLFLOAT	2-35
ROW, Named	Stores a named row type	2-35
SERIAL	Stores sequential integers in same range as INT	2-39
SERIAL8	Stores large sequential integers in same range as INT8	2-40
SMALLFLOAT	Stores single-precision floating-point numbers corresponding to the **float** data type in C	2-43

(2 of 3)

Data Type	Explanation	Page
SMALLINT	Stores whole numbers from –32,767 to +32,767	2-43
TEXT	Stores any kind of text data, up to 2^{31} bytes in length	2-44
VARCHAR(*m*,*r*)	Stores character strings of varying length (up to 255 bytes); collation is in code-set order.	2-46

(3 of 3)

For the character data types (CHAR, CHAR VARYING, NCHAR, NVARCHAR, and VARCHAR), a data string can include letters, digits, punctuation, whitespace, diacritical marks, ligatures, and other printable symbols from the code set of the database locale. (For some East Asian locales, multibyte characters are supported within data strings.)

IDS

Dynamic Server also supports the data types that Figure 2-3 lists. This chapter describes each of these data types.

Figure 2-3
Additional Data Types That Dynamic Server Supports

Data Type	Explanation	Page
BLOB	Stores binary data in random-access chunks	2-9
BOOLEAN	Stores Boolean values `true` and `false`	2-10
CLOB	Stores text data in random-access chunks	2-14
Distinct	Stores data in a user-defined type that has the same format as a source type on which it is based, but whose functions and casts can differ from those on the source type	2-23
INT8	Stores 8-byte integer values in range $-(2^{63} - 1)$ to $2^{63} - 1$	2-25
LIST(*e*)	Stores a sequentially ordered collection of elements, all of the same data type, *e*; allows duplicate values	2-29
LVARCHAR	Stores variable-length data that can exceed 255 bytes	2-30
MULTISET(*e*)	Stores a non-ordered collection of values, with elements all of the same data type, *e*; allows duplicate values.	2-32

(1 of 2)

Data Type	Explanation	Page
Opaque	Stores a user-defined data type whose internal structure is inaccessible to the database server	2-34
ROW, Named	Stores a named row type	2-35
ROW, Unnamed	Stores an unnamed row type	2-37
SERIAL8	Stores large sequential integers in same range as INT8	2-40
SET(*e*)	Stores a non-ordered collection of elements, all of the same data type, *e*; does not allow duplicate values	2-41

(2 of 2)

♦

For information about Informix internal data types that SQL statements support (such as IMPEX, IMPEXBIN, and SENDRECV), see *Creating User-Defined Routines and User-Defined Data Types*.

Description of Data Types

This section describes the data types that Informix database servers support. Icons mark the data types that only Dynamic Server supports.

IDS

BLOB

The BLOB data type stores any kind of binary data in random-access chunks, called sbspaces. Binary data typically consists of saved spreadsheets, program-load modules, digitized voice patterns, and so on. The database server performs no interpretation on the contents of a BLOB column. A BLOB column can be up to 4 terabytes in length.

The term *smart large object* refers to BLOB and CLOB data types. Use CLOB data types (see page 2-14) for random access to text data. For general information about BLOB and CLOB data types, see “Smart Large Objects” on page 2-50.

You can use the following SQL functions to perform some operations on a BLOB column:

- **FILETOBLOB** copies a file into a BLOB column.
- **LOTOFILE** copies a BLOB (or CLOB) value into an operating-system file.
- **LOCOPY** copies an existing smart large object to a new smart large object.

For more information on these SQL functions, see the *Informix Guide to SQL: Syntax*.

No casts exist for BLOB data. Therefore, the database server cannot convert data of type BLOB to any other data type. Within SQL, you are limited to the equality (=) comparison operation for BLOB data. To perform additional operations, you must use one of the application programming interfaces (APIs) from within your client application.

You can insert data into BLOB columns in the following ways:

- With the **dbload** or **onload** utilities
- With the LOAD statement (DB-Access)
- With the **FILETOBLOB** function
- From BLOB (**ifx_lo_t**) host variables (Informix ESQL/C)

If you select a BLOB column using DB-Access, only the string `<SBlob value>` is returned; no actual value is displayed.

IDS

BOOLEAN

The BOOLEAN data type stores true/false data as a single byte. The following table shows internal and literal representations of the BOOLEAN data type.

Logical Value	Internal Representation	Literal Representation
TRUE	\0	't'
FALSE	\1	'f'
NULL	Internal Use Only	NULL

You can compare two BOOLEAN values to test for equality or inequality. You can also compare a BOOLEAN value to the Boolean literals 't' and 'f'. BOOLEAN values are case insensitive; 't' is equivalent to 'T' and 'f' to 'F'.

You can use a BOOLEAN column to store what a Boolean expression returns. In the following example, the value of **boolean_column** is 't' if **column1** is less than **column2**, 'f' if **column1** is greater than or equal to **column2**, and NULL if the value of either **column1** or **column2** is unknown:

```
UPDATE my_table SET boolean_column = (column1 < column2)
```

BYTE

The BYTE data type stores any kind of binary data in an undifferentiated byte stream. Binary data typically consists of digitized information, such as spreadsheets, program load modules, digitized voice patterns, and so on.

The term *simple large objec*t refers to BYTE and TEXT data types.

The BYTE data type has no maximum size. A BYTE column has a theoretical limit of 2^{31} bytes and a practical limit that your disk capacity determines.

You can store, retrieve, update, or delete the contents of a BYTE column. You cannot, however, use BYTE operands in arithmetic or string operations, nor assign literals to BYTE columns with the SET clause of the UPDATE statement. You also cannot use BYTE items in any of the following ways:

- With aggregate functions
- With the IN clause
- With the MATCHES or LIKE clauses
- With the GROUP BY clause
- With the ORDER BY clause

BYTE operands are valid in Boolean expressions only when you are testing for NULL values with the IS NULL or IS NOT NULL operators.

You can insert data into BYTE columns in the following ways:

- With the **dbload** or **onload** utilities
- With the LOAD statement (DB-Access)
- From BYTE host variables (Informix ESQL/C)

You cannot use a quoted text string, number, or any other actual value to insert or update BYTE columns.

When you select a BYTE column, you can choose to receive all or part of it. To retrieve it all, use the regular syntax for selecting a column. You can also select any part of a BYTE column by using subscripts, as the following example shows:

```
SELECT cat_picture [1,75] FROM catalog WHERE catalog_num = 10001
```

This statement reads the first 75 bytes of the **cat_picture** column associated with the catalog number 10001.

The database server provides a cast to convert BYTE values to BLOB values. For more information, see the *Informix Guide to Database Design and Implementation*.

If you select a BYTE column using the DB-Access Interactive Schema Editor, only the string "`<BYTE value>`" is returned; no data value is displayed.

Important: *If you try to return a BYTE column from a subquery, you get an error message even when the BYTE column is not used in a comparison condition or with the IN predicate.*

CHAR(n)

The CHAR data type stores any sequence of letters, numbers, and symbols. It can store single-byte and multibyte characters, based on the database locale. (For more information on East Asian locales that support multibyte code sets, see "Multibyte Characters with CHAR" on page 2-14.)

A CHAR(*n*) column has a length *n* bytes, where 1 £ *n* £ 32,767. If you do not specify *n*, CHAR(1) is the default length. Character columns typically store alphanumeric strings, such as names, addresses, phone numbers, and so on.

When a value is retrieved or stored as CHAR(*n*), exactly *n* bytes of data are transferred. If the string is shorter than *n* bytes, the string is extended with blank spaces up to the declared length. If the data value is longer than *n* bytes, a data string of length *n* that has been truncated from the right is inserted or retrieved, without the database server raising an exception.

GLS

This does not create partial characters in multibyte locales. In right-to-left locales, such as Arabic, Hebrew, or Farsi, the truncation is from the left. ♦

Treating CHAR Values as Numeric Values

If you plan to perform calculations on numbers stored in a column, you should assign a number data type to that column. Although you can store numbers in CHAR columns, you might not be able to use them in some arithmetic operations. For example, if you insert a sum into a CHAR column, you might experience overflow problems if the CHAR column is too small to hold the value. In this case, the INSERT fails. Numbers that have leading zeros (such as some zip codes) have the zeros stripped if they are stored as number types INTEGER or SMALLINT. Instead, store these numbers in CHAR columns.

Sorting and Relational Comparisons

CHAR values are compared to other CHAR values by taking the shorter value and padding it on the right with blank spaces until the values have equal length. Then the two values are compared for the full length. Comparisons use the code-set collation order.

Nonprintable Characters with CHAR

A CHAR value can include tab, newline, whitespace, and nonprintable characters. You must, however, use an application to INSERT nonprintable characters into host variables and the host variables into your database. After passing nonprintable characters to the database server, you can store or retrieve them. After you SELECT nonprintable characters, FETCH them into host variables and display them with your own display mechanism.

If you try to display nonprintable characters with DB-Access, your screen returns inconsistent results. (Which characters are nonprintable is locale-dependent. See also the discussion of code-set conversion between the client and the database server in the *Informix Guide to GLS Functionality*.)

GLS

Collating CHAR Values

In general, the collation order of CHAR values is the order of characters as they appear in the code set. (An exception is the MATCHES operator with ranges; see “Collating VARCHAR Values” on page 2-47.) For more information about collation order, see the *Informix Guide to GLS Functionality*.

GLS

Multibyte Characters with CHAR

For multibyte code sets, the database locale must support any multibyte characters that a database uses. If you are storing multibyte characters, make sure to calculate the number of bytes needed. For more information on multibyte characters and locales, see the *Informix Guide to GLS Functionality*.

CHARACTER(n)

The CHARACTER data type is a synonym for CHAR.

CHARACTER VARYING(m,r)

The CHARACTER VARYING data type stores a string of letters, digits, and symbols of varying length, where *m* is the maximum size of the column and *r* is the minimum amount of space reserved for that column.

The CHARACTER VARYING data type complies with ANSI/ISO standard for SQL; the non-ANSI Informix VARCHAR data type supports the same functionality. See the description of the VARCHAR data type in "VARCHAR(m,r)" on page 2-46.

IDS

CLOB

The CLOB data type stores any kind of text data in random-access chunks, called sbspaces. Text data can include text-formatting information, as long as this information is also textual, such as PostScript, Hypertext Markup Language (HTML), Standard Graphic Markup Language (SGML), or Extensible Markup Language (XML) data.

The term *smart large object* refers to CLOB and BLOB data types. The CLOB data type supports special operations for character strings that are inappropriate for BLOB values. A CLOB value can be up to 4 terabytes in length.

Use the BLOB data type (see "BLOB" on page 2-9) for random access to binary data. For general information about the CLOB and BLOB data types, see "Smart Large Objects" on page 2-50.

You can use the following SQL functions to perform some operations on a CLOB column:

- **FILETOCLOB** copies a file into a CLOB column.
- **LOTOFILE** copies a CLOB (or BLOB) value into an operating-system file.
- **LOCOPY** copies an existing smart large object to a new smart large object.

For more information on these SQL functions, see the *Informix Guide to SQL: Syntax*.

No casts exist for CLOB data. Therefore, the database server cannot convert data of the CLOB type to any other data type. Within SQL, you are limited to the equality (=) comparison operation for CLOB data. To perform additional operations, you must use one of the application programming interfaces from within your client application.

Multibyte Characters with CLOB

You can insert data into CLOB columns in the following ways:

- With the **dbload** or **onload** utilities
- With the LOAD statement (DB-Access)
- From CLOB (**ifx_lo_t**) host variables (Informix ESQL/C)

For more information and examples for using the CLOB data type, see the *Informix Guide to SQL: Tutorial* and the *Informix Guide to Database Design and Implementation*.

GLS

With GLS, the following rules apply:

- Multibyte CLOB characters must be supported by the database locale.
- The CLOB data type is collated in code-set order.
- For CLOB columns, the database server handles any required code-set conversions for the data.

For more information on database locales, collation order, and code-set conversion, see the *Informix Guide to GLS Functionality*. ♦

DATE

The DATE data type stores the calendar date. DATE data types require 4 bytes. A calendar date is stored internally as an integer value equal to the number of days since December 31, 1899.

Because DATE values are stored as integers, you can use them in arithmetic expressions. For example, you can subtract a DATE value from another DATE value. The result, a positive or negative INTEGER value, indicates the number of days that elapsed between the two dates. (You can use a UNITS DAY expression to convert the result to an INTERVAL DAY TO DAY data type.)

The following example shows the default display format of a DATE column:

```
mm/dd/yyyy
```

In this example, *mm* is the month (1-12), *dd* is the day of the month (1-31), and *yyyy* is the year (0001-9999). You can specify a different order of time units and a different time-unit separator than `/` (or no separator) by setting the **DBDATE** environment variable; see “DBDATE” on page 3-32.

If you enter only a 2-digit value for the year, Informix products expand the year to 4 digits. If you enter the year as `99`, whether this is interpreted as `1999` or as `2099` depends on the **DBCENTURY** environment variable setting and the system clock-calendar. If you do not set **DBCENTURY**, Informix products use the leading digits of the current year to expand abbreviated years. For information on **DBCENTURY**, refer to page 3-28.

GLS

In non-default locales, you can display dates in culture-specific formats. The locale and the **GL_DATE** and **DBDATE** environment variables (as described in the next chapter) affect the display formatting of DATE values. They do not, however, affect the internal storage format for DATE columns in the database. For more information, see the *Informix Guide to GLS Functionality*. ♦

DATETIME

The DATETIME data type stores an instant in time expressed as a calendar date and time of day. You choose how precisely a DATETIME value is stored; its precision can range from a year to a fraction of a second.

The DATETIME data type is composed of a contiguous sequence of fields that represents each time unit (*year*, *month*, *day*, and so forth) that you want to record. Qualifiers to specify a DATETIME data type have this format:

```
DATETIME largest_qualifier TO smallest_qualifier
```

This resembles an INTERVAL data type qualifier (“INTERVAL” on page 2-26), but DATETIME represents a point in time, rather than (like INTERVAL) a span of time. The following differences exist between DATETIME and INTERVAL qualifiers:

- The DATETIME keyword replaces the INTERVAL keyword.
- DATETIME qualifiers cannot specify a non-default precision for the *largest_qualifier* time unit.
- A DATETIME value that includes YEAR and / or MONTH time units can also include smaller time units.

The *largest_qualifier* and *smallest_qualifier* of a DATETIME data type can be any of the fields that Figure 2-4 lists, provided that *smallest_qualifier* does not specify a larger time unit than *largest_qualifier*. (The two time units can be the same; for example, DATETIME YEAR TO YEAR.)

Figure 2-4
DATETIME Field Qualifiers

Qualifier Field	Valid Entries
YEAR	A year numbered from 1 to 9,999 (A.D.)
MONTH	A month numbered from 1 to 12
DAY	A day numbered from 1 to 31, as appropriate to the month
HOUR	An hour numbered from 0 (midnight) to 23

(1 of 2)

Qualifier Field	Valid Entries
MINUTE	A minute numbered from 0 to 59
SECOND	A second numbered from 0 to 59
FRACTION	A decimal fraction-of-a-second with up to 5 digits of scale. The default scale is 3 digits (a thousandth of a second). For *smallest_qualifier to* specify another scale, write FRACTION(*n*), where *n* is the desired number of digits from 1 to 5.

(2 of 2)

A DATETIME column does not need to include all fields from YEAR to FRACTION; it can include a subset of fields or even a single field. For example, you can enter a value of MONTH TO HOUR in a column that is defined as YEAR TO MINUTE, as long as each entered value contains information for a contiguous sequence of fields. You cannot, however, define a column for just MONTH and HOUR; this entry must also include a value for DAY.

If you use the DB-Access TABLE menu, and you do not specify the DATETIME qualifiers, the default DATETIME qualifier, YEAR TO YEAR, is assigned.

A valid DATETIME literal must include the DATETIME keyword, the values to be entered, and the field qualifiers. You must include these qualifiers because, as noted earlier, the value that you enter can contain fewer fields than were declared for that column. Acceptable qualifiers for the first and last fields are identical to the list of valid DATETIME fields that Figure 2-4 lists.

Write values for the field qualifiers as integers and separate them with delimiters. Figure 2-5 lists the delimiters that are used with DATETIME values in the default U.S. English locale. (These are a superset of the delimiters that are used in INTERVAL values; see “INTERVAL Delimiters” on page 2-28.)

Figure 2-5
Delimiters Used with DATETIME

Delimiter	Placement in DATETIME Literal
Hyphen (**-**)	Between the YEAR, MONTH, and DAY time-unit values
Blank space ( )	Between the DAY and HOUR time-unit values
Colon (**:**)	Between the HOUR, MINUTE, and SECOND time unit values
Decimal point (**.**)	Between the SECOND and FRACTION time-unit values

Figure 2-6 shows a DATETIME YEAR TO FRACTION(3) value with delimiters.

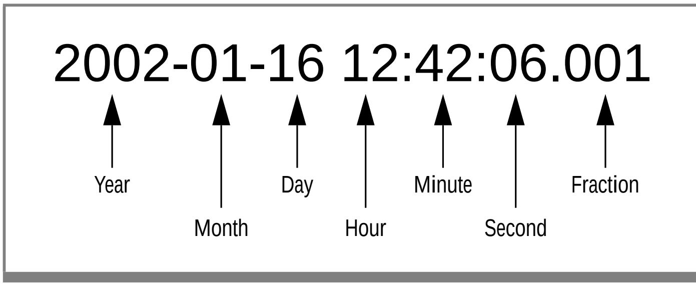


Figure 2-6
Example DATETIME Value with Delimiters

When you enter a value with fewer time-unit fields than in the column, the value that you enter is expanded automatically to fill all the declared time-unit fields. If you leave out any more significant fields, that is, time units larger than any that you include, those fields are filled automatically with the current values for those time units from the system clock-calendar. If you leave out any less-significant fields, those fields are filled with zeros (or with 1 for MONTH and DAY) in your entry.

You can also enter DATETIME values as character strings. The character string must include information for each field defined in the DATETIME column. The INSERT statement in the following example shows a DATETIME value entered as a character string:

```
INSERT INTO cust_calls (customer_num, call_dtime, user_id,
      call_code, call_descr)
   VALUES (101, '2001-01-14 08:45', 'maryj', 'D',
      'Order late - placed 6/1/00')
```

If **call_dtime** is declared as DATETIME YEAR TO MINUTE, the character string must include values for the *year*, *month*, *day*, *hour*, and *minute* fields.

If the character string does not contain information for all the declared fields (or if it adds additional fields), then the database server returns an error.

All fields of a DATETIME column are two-digit numbers except for the *year* and *fraction* fields. The *year* field is stored as four digits. When you enter a two-digit value in the year field, how the abbreviated year is expanded to four digits depends on the setting of the **DBCENTURY** environment variable.

For example, if you enter `02` as the *year* value, whether the year is interpreted as `1902`, `2002`, or `2102` depends on the setting of **DBCENTURY** and on the value of the system clock-calendar at execution time. If you do not set **DBCENTURY**, then the leading digits of the current year are appended by default. For information about setting **DBCENTURY**, see "DBCENTURY" on page 3-28.

The *fraction* field requires *n* digits where $1 \le n \le 5$, rounded up to an even number. You can use the following formula (rounded up to a whole number of bytes) to calculate the number of bytes that a DATETIME value requires:

```
(total number of digits for all fields) /2 + 1
```

For example, a YEAR TO DAY qualifier requires a total of eight digits (four for *year*, two for *month*, and two for *day*). According to the formula, this data value requires 5, or (8/2) + 1, bytes of storage.

For information on how to use DATETIME values in arithmetic and relational expressions, see "Manipulating DATE with DATETIME and INTERVAL Values" on page 2-54. For more information on the DATETIME data type, see the *Informix Guide to SQL: Syntax* and the *Informix Guide to GLS Functionality*.

GLS

If you specify a locale other than U.S. English, the locale defines the culture-specific display formats for DATETIME values. To change the default display format, change the setting of the **GL_DATETIME** environment variable.

With an ESQL API, the **DBTIME** environment variable also affects DATETIME formatting. Non-default locales and settings of the **GL_DATE** and **DBDATE** environment variables also affect the display of DATETIME data. They do not, however, affect the internal storage format of a DATETIME column.

The USEOSTIME configuration parameter can affect the subsecond granularity when the database server obtains the current time from the operating system in SQL statements; for details, see the *Administrator's Reference.*

For more information on **DBTIME**, see "DBTIME" on page 3-48. For more information on **DBCENTURY**, see "DBCENTURY" on page 3-28. For more information on locales and GLS environment variables, see the *Informix Guide to GLS Functionality*. ♦

DEC

The DEC data type is a synonym for DECIMAL.

DECIMAL

The DECIMAL data type can take two forms: DECIMAL(*p*) floating point and DECIMAL(*p,s*) fixed point. In an ANSI-compliant database, however, all DECIMAL numbers are fixed point.

By default, literal numbers that include a decimal (**.**) point are interpreted by the database server as DECIMAL values.

DECIMAL Floating Point

The DECIMAL data type stores decimal floating-point numbers up to a maximum of 32 significant digits, where *p* is the total number of significant digits (the *precision*). Specifying precision is optional. If you do not specify the precision (*p*), DECIMAL is treated as DECIMAL(16), a floating-point decimal with a precision of 16 places. DECIMAL(*p*) has an absolute exponent range between 10^{-130} and 10^{124}.

If you use an ANSI-compliant database and specify DECIMAL(*p*), the value defaults to DECIMAL(*p*, 0), meaning that only whole-number values can be stored in this data type.

DECIMAL Fixed Point

In fixed-point numbers, DECIMAL(*p,s*), the decimal point is fixed at a specific place, regardless of the value of the number. When you specify a column of this type, you declare its precision (*p*) as the total number of digits that it can store, from 1 to 32. You declare its *scale* (*s*) as the total number of digits in the fractional part (that is, to the right of the decimal point).

All numbers with an absolute value less than $0.5 * 10^{-s}$ have the value zero. The largest absolute value of a DECIMAL(*p,s*) data type that you can store without an overflow error is $10^{p-s} - 10^{-s}$. A DECIMAL column typically stores numbers with fractional parts that must be stored and displayed exactly (for example, rates or percentages). In an ANSI-compliant database, all DECIMAL numbers must have absolute values in the range 10^{-32} to 10^{+31}.

DECIMAL Storage

The database server uses one byte of disk storage to store two digits of a decimal number, plus an additional byte to store the exponent and sign. The significant digits to the left of the decimal and the significant digits to the right of the decimal are stored in separate groups of bytes. How the database server stores DECIMAL numbers is illustrated in the following example.

If you specify DECIMAL(6,3), the data type consists of three significant digits in the integral part and three significant digits in the fractional part (for instance, 123.456). The three digits to the left of the decimal are stored on 2 bytes (where one of the bytes only holds a single digit) and the three digits to the right of the decimal are stored on another 2 bytes, as Figure 2-7 illustrates.

(The exponent byte is not shown.) With the additional byte required for the exponent and sign, DECIMAL(6,3) requires a total of 5 bytes of storage.

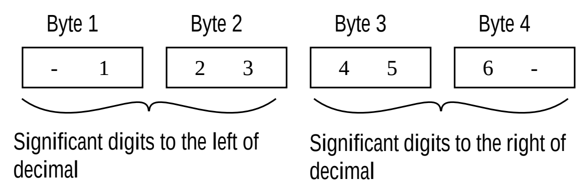


Figure 2-7
Schematic That Illustrates the Storage of Digits in a Decimal Value

You can use the following formulas (rounded down to a whole number of bytes) to calculate the byte storage (N) for a decimal data type (where N includes the byte that is required to store the exponent and the sign):

```
If the scale is odd: N = (precision + 4) / 2
If the scale is even: N = (precision + 3) / 2
```

For example, the data type DECIMAL(5,3) requires 4 bytes of storage (9/2 rounded down equals 4).

There is one caveat to these formulas. The maximum number of bytes the database server uses to store a decimal value is 17. One byte is used to store the exponent and sign, leaving 16 bytes to store up to 32 digits of precision. If you specify a precision of 32 and an *odd* scale, however, you lose 1 digit of precision. Consider, for example, the data type DECIMAL(32,31). This decimal is defined as 1 digit to the left of the decimal and 31 digits to the right. The 1 digit to the left of the decimal requires 1 byte of storage. This leaves only 15 bytes of storage for the digits to the right of the decimal. The 15 bytes can accommodate only 30 digits, so 1 digit of precision is lost.

IDS

Distinct

A *distinct* type is a data type that is derived from one of the following source types (called the *base type*):

- A built-in type
- An existing distinct type
- An existing named row type
- An existing opaque type

A DISTINCT type inherits from its source type the length and alignment on the disk. A DISTINCT type thus makes efficient use of the preexisting functionality of the database server.

When you create a DISTINCT data type, the database server automatically creates two explicit casts: one cast from the DISTINCT type to its source type and one cast from the source type to the DISTINCT type. A DISTINCT type based on a built-in source type does not inherit the built-in casts that are provided for the built-in type. A DISTINCT type does inherit, however, any user-defined casts that have been defined on the source type.

A DISTINCT type cannot be compared directly to its source type. To compare the two types, you must first explicitly cast one type to the other.

You must define a DISTINCT type in the database. Definitions of DISTINCT types are stored in the **sysxtdtypes** system catalog table. The following SQL statements maintain the definitions of DISTINCT types in the database:

- The CREATE DISTINCT TYPE statement adds a DISTINCT type to the database.
- The DROP TYPE statement removes a previously defined DISTINCT type from the database.

For more information about the SQL statements mentioned above, see the *Informix Guide to SQL: Syntax*. For information about casting DISTINCT data types, see “Casts for Distinct Types” on page 2-68. For examples that show how to create and register cast functions for a DISTINCT type, see the *Informix Guide to Database Design and Implementation*.

DOUBLE PRECISION

The DOUBLE PRECISION keywords are a synonym for the FLOAT keyword.

FLOAT(n)

The FLOAT data type stores double-precision floating-point numbers with up to 17 significant digits. FLOAT corresponds to IEEE 4-byte floating-point, and to the **double** data type in C. The range of values for the FLOAT data type is the same as the range of the C **double** data type on your computer.

You can use *n* to specify the precision of a FLOAT data type, but SQL ignores the precision. The value *n* must be a whole number between 1 and 14.

A column with the FLOAT data type typically stores scientific numbers that can be calculated only approximately. Because floating-point numbers retain only their most significant digits, the number that you enter in this type of column and the number the database server displays can differ slightly.

The difference between the two values depends on how your computer stores floating-point numbers internally. For example, you might enter a value of 1.1000001 into a FLOAT field and, after processing the SQL statement, the database server might display this value as 1.1. This situation occurs when a value has more digits than the floating-point number can store. In this case, the value is stored in its approximate form with the least significant digits treated as zeros.

FLOAT data types usually require 8 bytes of storage per value.

Conversion of a FLOAT value to a DECIMAL value results in 17 digits of precision.

INT

The INT data type is a synonym for INTEGER.

IDS

INT8

The INT8 data type stores whole numbers that can range in value from –9,223,372,036,854,775,807 to 9,223,372,036,854,775,807 [or $-(2^{63}-1)$ to $2^{63}-1$], for 18 or 19 digits of precision. The number –9,223,372,036,854,775,808 is a reserved value that cannot be used. The INT8 data type is typically used to store large counts, quantities, and so on.

The way that the database server stores the INT8 data is platform-dependent. On 64-bit platforms, INT8 is stored as a signed binary integer; the data type requires 8 bytes per value. On 32-bit platforms, the database server uses an internal format that can require up to 10 bytes of storage.

Arithmetic operations and sort comparisons are performed more efficiently on integer data than on floating-point or fixed-point decimal data, but INT8 cannot store data with absolute values beyond $|2^{63}-1|$. If a value exceeds the numeric range of INT8, the database server does not store the value.

INTEGER

The INTEGER data type stores whole numbers that range from –2,147,483,647 to 2,147,483,647, for 9 or 10 digits of precision. The number –2,147,483,648 is a reserved value and cannot be used. The INTEGER value is stored as a signed binary integer and is typically used to store counts, quantities, and so on.

Arithmetic operations and sort comparisons are performed more efficiently on integer data than on float or decimal data. INTEGER columns, however, cannot store absolute values beyond (2^{31}-1). If a data value lies outside the numeric range of INTEGER, the database server does not store the value.

INTEGER data types require 4 bytes of storage per value.

INTERVAL

The INTERVAL data type stores a value that represents a span of time. INTERVAL types are divided into two classes: *year-month intervals* and *day-time intervals*. A year-month interval can represent a span of years and months, and a day-time interval can represent a span of days, hours, minutes, seconds, and fractions of a second.

An INTERVAL value is always composed of one value, or a contiguous sequence of values, that represents time units. Within a data-definition statement such as CREATE TABLE or ALTER TABLE that defines the precision of an INTERVAL data type, the qualifiers must have the following format:

```
INTERVAL largest_qualifier(n) TO smallest_qualifier
```

Here the *largest_qualifier* and *smallest_qualifier* keywords are taken from one of the two INTERVAL classes, as shown in Figure 2-8 on page 2-26.

If SECOND (or a larger time unit) is the *largest_qualifier*, the declaration of an INTERVAL data type can optionally specify *n*, the precision of the largest time unit (for *n* ranging from 1 to 9); this is not a feature of DATETIME data types.

If *smallest_qualifier* is FRACTION, you can also specify a scale in the range from 1 to 5. For FRACTION TO FRACTION qualifiers, the upper limit of *n* is 5, rather than 9. There are two incommensurable classes of INTERVAL data types:

- Those with a *smallest_qualifier* larger than DAY
- Those with a *largest_qualifier* smaller than MONTH

Figure 2-8
Interval Classes

Interval Class	Time Units	Valid Entry
YEAR-MONTH INTERVAL	YEAR	A number of years
	MONTH	A number of months
DAY-TIME INTERVAL	DAY	A number of days
	HOUR	A number of hours
	MINUTE	A number of minutes
	SECOND	A number of seconds
	FRACTION	A decimal fraction of a second, with up to 5 digits. The default scale is 3 digits (thousandth of a second). To specify a non-default scale, write FRACTION(*n*), where *n* is the desired number of digits from 1 to 5.

As with DATETIME data types, you can define an INTERVAL to include only the subset of time units that you need. But because the construct of "month" (as used in calendar dates) is not a time unit that has a fixed number of days, a single INTERVAL value cannot combine months and days; arithmetic that involves operands of the two different INTERVAL classes is not supported.

A value entered into an INTERVAL column need not include the full range of time units that were specified in the data-type declaration of the column. For example, you can enter a value of HOUR TO SECOND precision into a column defined as DAY TO SECOND. A value must always consist, however, of a contiguous sequence of time units. In the previous example, you cannot enter only the HOUR and SECOND values; you must also include MINUTE values.

A valid INTERVAL literal contains the INTERVAL keyword, the values to be entered, and the field qualifiers. (See the discussion of literal intervals in the *Informix Guide to SQL: Syntax*.) When a value contains only one field, the largest and smallest fields are the same.

When you enter a value in an INTERVAL column, you must specify the largest and smallest fields in the value, just as you do for DATETIME values. In addition, you can optionally specify the precision of the first field (and the scale of the last field if it is a FRACTION). If the largest and smallest field qualifiers are both FRACTION, you can specify only the scale in the last field. Acceptable qualifiers for the largest and smallest fields are identical to the list of INTERVAL fields that Figure 2-8 on page 2-26 displays.

If you use the DB-Access TABLE menu, and you do not specify the INTERVAL field qualifiers, the default INTERVAL qualifier, YEAR TO YEAR, is assigned.

The *largest_qualifier* in an INTERVAL value can be up to nine digits (except for FRACTION, which cannot be more than five digits), but if the value that you want to enter is greater than the default number of digits allowed for that field, you must explicitly identify the number of significant digits in the value that you enter. For example, to define an INTERVAL of DAY TO HOUR that can store up to 999 days, you could specify it the following way:

```
INTERVAL DAY(3) TO HOUR
```

INTERVAL literals use the same delimiters as DATETIME literals (except that MONTH and DAY time units are not valid within the same INTERVAL value). Figure 2-9 shows the INTERVAL delimiters.

Figure 2-9
INTERVAL Delimiters

Delimiter	Placement in DATETIME Expression
Hyphen	Between the YEAR and MONTH portions of the value
Blank space	Between the DAY and HOUR portions of the value
Colon	Between the HOUR, MINUTE, and SECOND portions of the value
Decimal point	Between the SECOND and FRACTION portions of the value

You can also enter INTERVAL values as character strings. The character string must include information for the same sequence of time units that was declared for the column. The INSERT statement in the following example shows an INTERVAL value entered as a character string:

```
INSERT INTO manufact (manu_code, manu_name, lead_time)
   VALUES ('BRO', 'Ball-Racquet Originals', '160')
```

Because the **lead_time** column is defined as INTERVAL DAY(3) TO DAY, this INTERVAL value requires only one field, the span of days required for lead time. If the character string does not contain information for all fields (or adds additional fields), the database server returns an error. For more information on entering INTERVAL values as character strings, see the *Informix Guide to SQL: Syntax*.

By default, all fields of an INTERVAL column are two-digit numbers, except for the year and fraction fields. The year field is stored as four digits. The fraction field requires *n* digits where 1 £ *n* £ 5, rounded up to an even number. You can use the following formula (rounded up to a whole number of bytes) to calculate the number of bytes required for an INTERVAL value:

```
(total number of digits for all fields)/2 + 1
```

For example, a YEAR TO MONTH qualifier requires a total of six digits (four for year and two for month). This data value requires 4, or (6/2) + 1, bytes of storage.

For information on using INTERVAL data in arithmetic and relational operations, see “Manipulating DATE with DATETIME and INTERVAL Values” on page 2-54. For information on using INTERVAL as a constant expression, see the description of the INTERVAL Field Qualifier in the *Informix Guide to SQL: Syntax*.

IDS

LIST(e)

The LIST data type is a collection type that stores ordered, non-unique elements; that is, it allows duplicate element values. The elements of a LIST have ordinal positions; That is, the list has a first element, a second element, and so on. (For a collection type with no ordinal positions, see the MULTISET data type in “MULTISET(e)” on page 2-32 and the SET data type in “SET(e)” on page 2-41.)

By default, the database server inserts LIST elements at the end of the list. To support the ordinal position of a LIST, the INSERT statement provides the AT clause. This clause allows you to specify the position at which you want to insert a list-element value. For more information, see the INSERT statement in the *Informix Guide to SQL: Syntax*.

All elements in a LIST have the same element type. To specify the element type, use the following syntax:

```
LIST(element_type NOT NULL)
```

The *element_type* of a LIST can be any of the following data types:

- A built-in type, except SERIAL, SERIAL8, BYTE, and TEXT
- A DISTINCT type
- An unnamed or named ROW type
- Another collection type
- An OPAQUE type

You must specify the NOT NULL constraint for LIST elements. No other constraints are valid for LIST columns. For more information on the syntax of the LIST data type, see the *Informix Guide to SQL: Syntax*.

You can use LIST anywhere that you would use any other data type, for example:

- After the IN predicate in the WHERE clause of a SELECT statement to search for matching LIST values
- As an argument to the CARDINALITY or **mi_collection_card( )** function to determine the number of elements in a LIST column

You *cannot* use LIST values as arguments to an aggregate function such as AVG, MAX, MIN, or SUM.

Two LIST values are equal if they have the same elements in the same order. The following examples both are LIST values but are not equal:

```
LIST{"blue", "green", "yellow"}
LIST{"yellow", "blue", "green"}
```

The above statements are not equal because the values are not in the same order. To be equal, the second statement would have to be:

```
LIST{"blue", "green", "yellow"}
```

IDS

LVARCHAR

You can use the LVARCHAR data type to create a column for storing variable-length character strings that can be larger than 255 bytes. By default, the database server interprets quoted strings as LVARCHAR data types.

The LVARCHAR data type is also used for input and output casts for opaque data types. The LVARCHAR data type stores opaque data types in the string (external) format. Each opaque type has an input support function and cast, which convert it from LVARCHAR to a form that database servers can manipulate. Each opaque type also has an output support function and cast, which convert it from its internal representation to LVARCHAR.

Important: *When LVARCHAR data is stored in a table column, the value is limited to 2 kilobytes (2Kb). When LVARCHAR is used in I/O operations on an opaque type, the size is limited only by the operating system.*

The LVARCHAR data type supports only a subset of the string operations that you can perform on CHAR and VARCHAR data types.

LVARCHAR is implemented as an opaque UDT. For more information about LVARCHAR, see *Creating User-Defined Routines and User-Defined Data Types*.

MONEY(p,s)

The MONEY data type stores currency amounts. Like the DECIMAL(*p*,*s*) data type, MONEY can store fixed-point numbers up to a maximum of 32 significant digits, where *p* is the total number of significant digits (the precision) and *s* is the number of digits to the right of the decimal point (the scale).

Unlike the DECIMAL data type, the MONEY data type is always treated as a fixed-point decimal number. The database server defines the data type MONEY(*p*) as DECIMAL(*p*,2). If the precision and scale are not specified, the database server defines a MONEY column as DECIMAL(16,2).

You can use the following formula (rounded down to a whole number of bytes) to calculate the byte storage for a MONEY data type:

```
If the scale is odd: N = (precision + 4) / 2
If the scale is even: N = (precision + 3) / 2
```

For example, a MONEY data type with a precision of 16 and a scale of 2 (MONEY(16,2)) requires 10 or (16 + 3)/2, bytes of storage.

In the default locale, client applications format values from MONEY columns with the following currency notation:

- A currency symbol: a dollar sign (**$**) at the front of the value
- A thousands separator: a comma (**,**) that separates every three digits in the integer part of the value
- A decimal point: a period (**.**) between the integer and fractional parts of the value

To change the format for MONEY values, change the **DBMONEY** environment variable. For information on how to set **DBMONEY**, see "DBMONEY" on page 3-38.

GLS

The default value that the database server uses for scale is locale-dependent. The default locale specifies a default scale of two. For non-default locales, if the scale is omitted from the declaration, the database server creates MONEY values with a locale-specific scale.

The currency notation that client applications use is locale-dependent. If you specify a nondefault locale, the client uses a culture-specific format for MONEY values that might differ from the default U.S. English format in the leading (or trailing) currency symbol, thousands separator, and decimal separator, depending on what the locale files specify. For more information on locale dependency, see the *Informix Guide to GLS Functionality*. ♦

IDS

MULTISET(e)

The MULTISET data type is a collection type that stores non-unique elements: it allows duplicate element values. The elements in a MULTISET have no ordinal position. That is, there is no concept of a first, second, or third element in a MULTISET. (For a collection type with ordinal positions for elements, see the LIST data type on page 2-29.)

All elements in a MULTISET have the same element type. To specify the element type, use the following syntax:

```
MULTISET(element_type NOT NULL)
```

The *element_type* of a collection can be any of the following types:

- A built-in type, except SERIAL, SERIAL8, BYTE, and TEXT
- An unnamed or a named row type
- Another collection type
- An opaque type

You must specify the NOT NULL constraint for MULTISET elements. No other constraints are valid for MULTISET columns. For more information on the syntax of the MULTISET collection type, see the *Informix Guide to SQL: Syntax*.

You can use MULTISET anywhere that you use any other data type, unless otherwise indicated. For example:

- After the IN predicate in the WHERE clause of a SELECT statement to search for matching MULTISET values
- As an argument to the CARDINALITY or **mi_collection_card( )** function to determine the number of elements in a MULTISET column

You *cannot* use MULTISET values as arguments to an aggregate function such as AVG, MAX, MIN, or SUM.

Two MULTISET data values are equal if they have the same elements, even if the elements are in different positions within the set. The following examples are both MULTISET values but are not equal:

```
MULTISET {"blue", "green", "yellow"}
MULTISET {"blue", "green", "yellow", "blue"}
```

The following MULTISET values are equal:

```
MULTISET {"blue", "green", "blue", "yellow"}
MULTISET {"blue", "green", "yellow", "blue"}
```

Named Row

See “Row, Named” on page 2-35.

GLS

NCHAR(n)

The NCHAR data type stores fixed-length character data. The data can be a sequence of single-byte or multibyte letters, digits, and other symbols. The main difference between CHAR and NCHAR data types is the collation order. While the collation order of the CHAR data type is defined by the code-set order, the collation order of the NCHAR data type depends on the locale-specific localized order. For more information about NCHAR, see the *Informix Guide to GLS Functionality*. See also the description of "DBNLS" on page 3-40.

NUMERIC(p,s)

The NUMERIC data type is a synonym for fixed-point DECIMAL.

GLS

NVARCHAR(m,r)

The NVARCHAR data type stores strings of varying lengths. The string can include digits, symbols, and single-byte and (in some locales) multibyte characters. The main difference between VARCHAR and NVARCHAR data types is the collation order. Collation of VARCHAR data follows code-set order, but NVARCHAR collation can be locale specific. (The section "Collating VARCHAR Values" on page 2-47 describes an exception.) For more information about NVARCHAR, see the *Informix Guide to GLS Functionality*.

IDS

Opaque

An opaque type is a data type for which you must provide the following information to the database server:

- A data structure for how the data values are stored on disk
- Support functions to determine how to convert between the disk storage format and the user format for data entry and display
- Secondary access methods that determine how the index on this data type is built, used, and manipulated
- User functions that use the data type
- A row in a system catalog table to register the opaque type in the database

The internal structure of an opaque type is not visible to the database server, and can only be accessed through user-defined routines. Definitions for opaque types are stored in the **sysxtdtypes** system catalog table. These SQL statements maintain the definitions of opaque types in the database:

- The CREATE OPAQUE TYPE statement adds an opaque type to the database.
- The DROP TYPE statement removes a previously defined opaque type from the database.

For more information on the above-mentioned SQL statements, see the *Informix Guide to SQL: Syntax*. For information on how to create opaque types and an example of an opaque type, see *Creating User-Defined Routines and User-Defined Data Types*.

REAL

The REAL data type is a synonym for SMALLFLOAT.

IDS

Row, Named

A named row type is declared by its name. That identifier must be unique within the schema. An unnamed row type is a row type that contains fields but has no user-defined name. Use a named row type if you want to use type inheritance. For more information, see "Row Data Types" on page 2-60.

Defining Named Row Types

You must define a named row type in the database. Definitions for named row types are stored in the **sysxtdtypes** system catalog table.

The fields of a row type can be any data type. The fields of a row type that are TEXT or BYTE type can be used in typed tables only. If you want to assign a row type to a column, its elements cannot be TEXT or BYTE data types.

In general, the data type of the field of a row type can be any of the following types:

- A built-in type (except for the TEXT or BYTE data types)
- A collection type (LIST, MULTISET, or SET)
- A distinct type
- A row type
- An opaque type

The following SQL statements maintain the definitions of named row types in the database:

- The CREATE ROW TYPE statement adds a named row type to the database.
- The DROP ROW TYPE statement removes a previously defined named row type from the database.

For details about these SQL syntax statements, see the *Informix Guide to SQL: Syntax*. For examples of how to create and use named row types, see the *Informix Guide to Database Design and Implementation*.

Equivalence and Named Row Types

No two named row types can be equal, even if they have identical structures, because they have different names. For example, the following named row types have the same structure (the same number of fields, and the same sequence of data types of fields within the row) but are not equal:

```
name_t (lname CHAR(15), initial CHAR(1), fname CHAR(15))
emp_t (lname CHAR(15), initial CHAR(1), fname CHAR(15))
```

Named Row Types and Inheritance

Named row types can be part of a type-inheritance hierarchy. That is, one named row type can be the parent (supertype) of another named row type. A subtype in a hierarchy inherits all the properties of its supertype. Type inheritance is discussed in the CREATE ROW TYPE statement in the *Informix Guide to SQL: Syntax* and in the *Informix Guide to Database Design and Implementation*.

Typed Tables

Tables that are part of an inheritance hierarchy must be typed tables. Typed tables are tables that have been assigned a named row type. For the syntax you use to create typed tables, see the CREATE TABLE statement in the *Informix Guide to SQL: Syntax*. Table inheritance and how it relates to type inheritance is also discussed in that section. For information about how to create and use typed tables, see the *Informix Guide to Database Design and Implementation*.

IDS

Row, Unnamed

An unnamed row type contains fields but has no user-defined name. An unnamed row type is defined by its structure. Two unnamed row types are equal if they have the same structure (meaning the ordered list of the data types of the fields). If two unnamed row types have the same number of fields, and if the sequence of the data type of each field in one row type matches the sequence of data types of the corresponding fields in the other ROW data type, then the two unnamed row data types are equal.

For example, the following unnamed row types are equal:

```
ROW (lname char(15), initial char(1) fname char(15))
ROW (dept char(15), rating char(1) name char(15))
```

The following row types have the same number of fields and the same data types, but are not equal, because their fields are not in the same order:

```
ROW (x integer, y varchar(20), z real)
ROW (x integer, z real, y varchar(20))
```

A field of an unnamed row type can be any of the following data types:

- A built-in type
- A collection type
- A distinct type
- A row type
- An opaque type

Unnamed row types cannot be used in typed tables or in type inheritance hierarchies.

For more information on unnamed row types, see the *Informix Guide to SQL: Syntax* and the *Informix Guide to Database Design and Implementation*.

Creating Unnamed Row Types

You can create an unnamed row type in several ways:

- You can declare an unnamed row type using the row keyword. Each field in a row can have a different field type. To specify the field type, use the following syntax:

  ```
  ROW(field_name field_type, ...)
  ```

  The *field_name* must conform to the rules for SQL identifiers. For more information, see the Identifier section in the *Informix Guide to SQL: Syntax*.
- You can generate an unnamed row type using row as a constructor and a series of values. A corresponding unnamed row type is created, using the default data types of the specified values.

  For example, a declaration of the following row value:

  ```
  ROW(1, 'abc', 5.30)
  ```

  defines this row type:

  ```
  ROW (x INTEGER, y VARCHAR, z DECIMAL)
  ```
- You can create an unnamed row type by an implicit or explicit cast from a named row type or from another unnamed row type.
- The rows of any table (except a table defined on a named row type) are unnamed row types.

Inserting Values into Unnamed Row Type Columns

When you specify field values for an unnamed row type, list the field values after the constructor and between parentheses. For example, suppose you have an unnamed row -type column. The following INSERT statement adds one group of field values to this row column:

```
INSERT INTO table1 VALUES (ROW(4, 'abc'))
```

You can specify a row column in the IN predicate in the WHERE clause of a SELECT statement to search for matching row values. For more information, see the Condition section in the *Informix Guide to SQL: Syntax*.

SERIAL(n)

The SERIAL data type stores a sequential integer that is automatically assigned by the database server when a new row is inserted. You can define only one SERIAL column in a table.

SERIAL values in a column are not automatically unique. You must apply a unique index or primary key constraint to this column to prevent duplicate SERIAL numbers. If you use the interactive schema editor in DB-Access to define the table, a unique index is applied automatically to a SERIAL column.

Also, SERIAL numbers might not be contiguous due to such factors as multiuser systems and rollbacks.

The DEFINE *variable* LIKE *column* syntax of SPL for indirect typing declares a variable of the INTEGER data type if *column* is a SERIAL data type.

The default SERIAL starting number is 1, but you can assign a non-default initial value, *n*, when you create or alter the table. Any number greater than 0 can be your starting number. The maximum SERIAL is 2,147,483,647. If you assign a number greater than 2,147,483,647, you receive a syntax error. (Use the SERIAL8 data type, rather than SERIAL, if you need a larger range.)

After a nonzero number is assigned, it cannot be changed. You can insert a value into a SERIAL column (using the INSERT statement) or reset a SERIAL column (using the ALTER TABLE statement), if the new value does not duplicate any existing value in the column. To insert into a SERIAL column, your database server increments by 1 the previous value (or the reset value, if that is larger) and assigns the result as the entered value. If ALTER TABLE has reset the next value of a SERIAL column to a value smaller than values already in that column, however, the next value follows this formula:

```
(maximum existing value in SERIAL column) + 1
```

For example, if you reset the SERIAL value of **customer.customer_num** to `50`, when the largest existing value is `128`, the next assigned number will be `129`. For more details on SERIAL data entry, see the *Informix Guide to SQL: Syntax*.

A SERIAL column can store unique codes (for example, order, invoice, or customer numbers). SERIAL data values require 4 bytes of storage, and have the same precision as the INTEGER data type. For details of another way to assign unique whole numbers to each row of a database table, see the CREATE SEQUENCE statement in *Informix Guide to SQL: Syntax*.

IDS

SERIAL8

The SERIAL8 data type stores a sequential integer assigned automatically by the database server when a new row is inserted. It behaves like the SERIAL data type, but with a larger range. (For more information on how to insert values into SERIAL8 columns, see the *Informix Guide to SQL: Syntax*.)

A SERIAL8 data column is commonly used to store large, unique numeric codes (for example, order, invoice, or customer numbers). SERIAL8 data values have the same precision and storage requirements as INT8 values (page 2-25). The following restrictions apply to SERIAL8 columns:

- You can define only one SERIAL8 column in a table.

  However, a table can have one SERIAL8 and one SERIAL column.
- SERIAL8 column values are not automatically unique.

  You must apply a unique index or primary key constraint to this column to prevent duplicate SERIAL8 numbers.
- The SERIAL8 data type does not allow a NULL value.

If you are using the interactive schema editor in DB-Access to define the table, a unique index is applied automatically to a SERIAL8 column.

The DEFINE *variable* LIKE *column* syntax of SPL for indirect typing declares a variable of the INT8 data type if *column* is a SERIAL8 data type.

Assigning a Starting Value for SERIAL8

The default serial starting number is 1, but you can assign an initial value, *n*, when you create or alter the table. To start the values at `1` in a SERIAL8 column of a table, give the value `0` for the SERIAL8 column when you insert rows into that table.The database server will assign the value `1` to the SERIAL8 column of the first row of the table. The highest SERIAL8 number you can assign is 2^{63}-1 (9,223,372,036,854,775,807). If you assign a number greater than this value, you receive a syntax error. When the database server generates a SERIAL8 value of this maximum number, it wraps around and starts generating values beginning at `1`.

After a nonzero number is assigned, it cannot be changed. You can, however, insert a value into a SERIAL8 column (using the INSERT statement) or reset the serial value *n* (using the ALTER TABLE statement), as long as that value does not duplicate any existing values in the table.

When you insert a number into a SERIAL8 column or reset the next value of a SERIAL8 column, your database server assigns the next number in sequence to the number entered. If you reset the next value of a SERIAL8 column to a value that is less than the values already in that column, however, the next value is computed using the following formula:

```
maximum existing value in SERIAL8 column + 1
```

For example, if you reset the serial value of the **customer_num** column in the **customer** table to 50, when the highest-assigned customer number is 128, the next customer number assigned is 129.

Using SERIAL8 with INT8

All the arithmetic operators that are valid for INT8 (such as +, -, *, and /) and all the SQL functions that are legal for INT8 (such as ABS, MOD, POW, and so on) are also valid for SERIAL8 values. Data conversion rules that apply to INT8 also apply to SERIAL8, but with a NOT NULL constraint on SERIAL8.

The value of a SERIAL8 column of one table can be stored in an INT8 columns of another table. In the second table, however, the INT8 values are not subject to the constraints on the original SERIAL8 column.

IDS

SET(e)

The SET data type is a collection type that stores unique elements; it does not allow duplicate element values. (For a collection type that does allow duplicate values, see the description of MULTISET in "MULTISET(e)" on page 2-32.)

The elements in a SET have no ordinal position. That is, no concept of a first, second, or third element in a SET exists. (For a collection type with ordinal positions for elements, see the LIST data type in "LIST(e)" on page 2-29.)

All elements in a SET have the same element type. To specify the element type, use the following syntax:

```
SET(element_type NOT NULL)
```

The *element_type* of a collection can be any of the following types:

- A built-in type, except SERIAL, SERIAL8, BYTE, and TEXT
- A named or unnamed row type
- Another collection type
- An opaque type

You must specify the NOT NULL constraint for SET elements. No other constraints are valid for SET columns. For more information on the syntax of the SET collection type, see the *Informix Guide to SQL: Syntax*.

You can use SET anywhere that you use any other data type, unless otherwise indicated. For example:

- After the IN predicate in the WHERE clause of a SELECT statement to search for matching SET values
- As an argument to the CARDINALITY or **mi_collection_card( )** function to determine the number of elements in a SET column

You *cannot* use SET values as arguments to an aggregate function such as AVG, MAX, MIN, or SUM.

For more information, see the Condition and Expression section in the *Informix Guide to SQL: Syntax*.

The following examples declare two sets. The first example declares a set of integers and the second declares a set of character elements.

```
SET(INTEGER NOT NULL)
SET(CHAR(20) NOT NULL)
```

The following examples construct the same sets from value lists:

```
SET{1, 5, 13}
SET{"Oakland", "Menlo Park", "Portland", "Lenexa"}
```

In the following example, a SET constructor function is part of a CREATE TABLE statement:

```
CREATE TABLE tab
(
   c CHAR(5),
   s SET(INTEGER NOT NULL)
);
```

The following SET values are equal:

```
SET{"blue", "green", "yellow"}
SET{"yellow", "blue", "green"}
```

SMALLFLOAT

The SMALLFLOAT data type stores single-precision floating-point numbers with approximately nine significant digits. SMALLFLOAT corresponds to the **float** data type in C. The range of values for a SMALLFLOAT data type is the same as the range of values for the C **float** data type on your computer.

A SMALLFLOAT data type column typically stores scientific numbers that can be calculated only approximately. Because floating-point numbers retain only their most significant digits, the number that you enter in this type of column and the number the database displays might differ slightly depending on how your computer stores floating-point numbers internally.

For example, you might enter a value of 1.1000001 in a SMALLFLOAT field and, after processing the SQL statement, the application development tool might display this value as 1.1. This difference occurs when a value has more digits than the floating-point number can store. In this case, the value is stored in its approximate form with the least significant digits treated as zeros.

SMALLFLOAT data types usually require 4 bytes per value.

Conversion of a SMALLFLOAT value to a DECIMAL value results in 9 digits of precision.

SMALLINT

The SMALLINT data type stores small whole numbers that range from –32,767 to 32,767. The maximum negative number, –32,768, is a reserved value and cannot be used. The SMALLINT value is stored as a signed binary integer.

Integer columns typically store counts, quantities, and so on. Because the SMALLINT data type takes up only 2 bytes per value, arithmetic operations are performed efficiently. However, this data type stores a limited range of values. If the values exceed the range between the minimum and maximum numbers, the database server does not store the value and provides you with an error message.

TEXT

The TEXT data type stores any kind of text data. It can contain both single-byte and multibyte characters that the locale supports.

The TEXT data type has no maximum size. A TEXT column has a theoretical limit of 2^{31} bytes and a practical limit that your available disk storage determines.

The term *simple large object* refers to the TEXT and BYTE data types.

TEXT columns typically store memos, manual chapters, business documents, program source files, and so on. In the default U.S. English locale, data objects of type TEXT can contain a combination of printable ASCII characters and the following control characters:

- Tab (CTRL-I)
- New line (CTRL-J)
- New page (CTRL-L)

You can store, retrieve, update, or delete the contents of a TEXT column. You cannot, however, use TEXT operands in arithmetic or string expressions, nor can you assign literals to TEXT columns with the SET clause of the UPDATE statement. You also cannot use TEXT items in any of the following ways:

- With aggregate functions
- With the IN clause
- With the MATCHES or LIKE clauses
- With the GROUP BY clause
- With the ORDER BY clause

You can use TEXT operands in Boolean expressions only when you are testing for NULL values with the IS NULL or IS NOT NULL operators.

You can insert data into TEXT columns in the following ways:

- With the **dbload** or **onload** utilities
- With the LOAD statement (DB-Access)
- From TEXT host variables (Informix ESQL/C)

You cannot use a quoted text string, number, or any other actual value to insert or update TEXT columns.

When you select a TEXT column, you can choose to receive all or part of it. To retrieve it all, use the regular syntax for selecting a column. You can also select any part of a TEXT column by using subscripts, as the following example shows:

```
SELECT cat_descr [1,75] FROM catalog WHERE catalog_num = 10001
```

This statement reads the first 75 bytes of the **cat_descr** column associated with the **catalog_num** value `10001`.

The database server provides a cast to convert TEXT objects to CLOB objects. For more information, see the *Informix Guide to Database Design and Implementation*.

Important: *If you try to return a TEXT column from a subquery, you get an error message even when the TEXT column is not used in a comparison condition or with the IN predicate.*

Nonprintable Characters with TEXT

Both printable and nonprintable characters can be inserted in TEXT columns. Informix products do not do any checking of data values that are inserted in a column of the TEXT data type. (Applications may have difficulty, however, in displaying TEXT values that include non-printable characters.) For detailed information on entering and displaying nonprintable characters, refer to "Nonprintable Characters with CHAR" on page 2-13.

Collating TEXT Data

The TEXT data type is collated in code-set order. For more information on collation orders, see the *Informix Guide to GLS Functionality*.

GLS

Multibyte Characters with TEXT

The database locale must support multibyte TEXT characters. For more information, see the *Informix Guide to GLS Functionality*.

Unnamed Row

See "Row, Unnamed" on page 2-37.

VARCHAR(m,r)

The VARCHAR data type stores character strings of varying length that contain single-byte and (if the locale supports them) multibyte characters, where *m* is the maximum size (in bytes) of the column and *r* is the minimum number of bytes reserved for that column.

The VARCHAR data type is the Informix implementation of a character varying data type. The ANSI standard data type for varying-length character strings is CHARACTER VARYING and is described in "CHARACTER VARYING(m,r)" on page 2-14.

You must specify the maximum size (*m*) of the VARCHAR column. The size of this parameter can range from 1 to 255 bytes. If you are placing an index on a VARCHAR column, the maximum size is 254 bytes. You can store character strings that are shorter, but not longer, than the *m* value that you specify.

Specifying the minimum reserved space (*r*) parameter is optional. This value can range from `0` to `255` bytes but must be less than the maximum size (*m*) of the VARCHAR column. If you do not specify any minimum value, it defaults to 0. You should specify this parameter when you initially intend to insert rows with short or NULL character strings in the column, but later expect the data to be updated with longer values.

Although the use of VARCHAR economizes on space used in a table, it has no effect on the size of an index. In an index based on a VARCHAR column, each index key has length *m*, the maximum size of the column.

When you store a VARCHAR value in the database, only its defined characters are stored. The database server does not strip a VARCHAR string of any user-entered trailing blanks, nor does the database server pad the VARCHAR to the full length of the column. However, if you specify a minimum reserved space (*r*) and some data values are shorter than that amount, some space reserved for rows goes unused.

VARCHAR values are compared to other VARCHAR values (and to other character-string data types) in the same way that CHAR values are compared. The shorter value is padded on the right with blank spaces until the values have equal lengths; then they are compared for the full length.

Nonprintable Characters with VARCHAR

Nonprintable VARCHAR characters are entered, displayed, and treated in the same way that nonprintable characters in CHAR values are treated. For details, see the section "Nonprintable Characters with CHAR" on page 2-13.

Storing Numeric Values in a VARCHAR Column

When you insert a numeric value in a VARCHAR column, the stored value does not get padded with trailing blanks to the maximum length of the column. The number of digits in a numeric VARCHAR value is the number of characters that you need to store that value. For example, given the following statement, the value that gets stored in table **mytab** is `1`.

```
create table mytab (col1 varchar(10));
insert into mytab values (1);
```

Tip: *VARCHAR treats C NULL (binary 0) and string terminators as termination characters for nonprintable characters.*

GLS

Multibyte Characters with VARCHAR

In some East Asian locales, VARCHAR data types can store multibyte characters if the database locale supports a multibyte code set. If you store multibyte characters, make sure to calculate the number of bytes needed. For more information, see the *Informix Guide to GLS Functionality*.

Collating VARCHAR Values

The main difference between the NVARCHAR and the VARCHAR data types (like the difference between CHAR and NCHAR) is the difference in collation sequencing. In general, the collation order of VARCHAR values (like CHAR values) is the order of the characters as they appear in the code set.

An exception is the MATCHES operator, which applies a localized collation order to both NVARCHAR and VARCHAR values (as well as to CHAR and to NCHAR values) when you use bracket ([ ]) symbols to specify ranges and the locale files define a non-default collation order. For more information, see the *Informix Guide to GLS Functionality*.

Built-In Data Types

Informix database servers support the following built-in data types.

Category	Data Types
Character	CHAR, CHARACTER VARYING, LVARCHAR, NCHAR, NVARCHAR, VARCHAR
Numeric	DECIMAL, FLOAT, INT8, INTEGER, MONEY, SERIAL, SERIAL8, SMALLFLOAT, SMALLINT
Large-object	Simple-large-object types: BYTE, TEXT Smart-large-object types: BLOB, CLOB
Time	DATE, DATETIME, INTERVAL
Miscellaneous	BOOLEAN

For a description of character, numeric, and miscellaneous data types, refer to the appropriate entry in “Description of Data Types” on page 2-9. Page references are in the alphabetical list in Figure 2-2 on page 2-6.

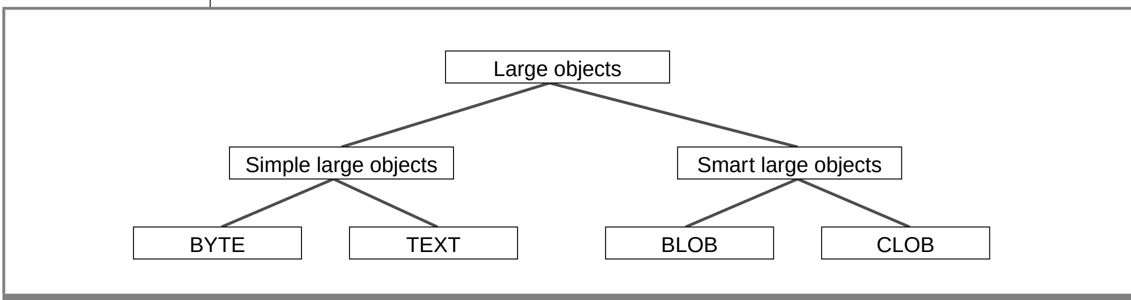
The following sections provide additional information on large-object and time data types.

Large-Object Data Types

A large object is a data object that is logically stored in a table column but physically stored independently of the column. Large objects are stored separately from the table because they typically store a large amount of data. Separation of this data from the table can increase performance.

Figure 2-10 shows the large-object data types.

Figure 2-10
Large-Object Data Types



IDS

Only Dynamic Server supports BLOB and CLOB data types. ♦

For the relative advantages and disadvantages of simple and smart large objects, see the *Informix Guide to Database Design and Implementation*.

Simple Large Objects

Simple large objects are a category of large objects that have a theoretical limit of 2^{31} bytes and a practical limit that your disk capacity determines. Informix database servers support the following simple-large-object data types:

BYTE — Stores binary data. For more detailed information about this data type, see the description on page 2-11.

TEXT — Stores text data. For more detailed information about this data type, see the description on page 2-44.

Unlike smart large objects, simple large objects do not support random access to the data. When you transfer a simple large object between a client application and the database server, you must transfer the entire BYTE or TEXT value. If the data does not fit into memory, you must store it in an operating-system file and then retrieve it from that file.

The database server stores simple large objects in *blobspaces*. A *blobspace* is a logical storage area that contains one or more chunks that only store BYTE and TEXT data. For information on how to define blobspaces, see your *Administrator's Guide*.

IDS

Smart Large Objects

Smart large objects are a category of large objects that support random access to the data and are generally recoverable. The random access feature allows you to seek and read through the smart large object as if it were an operating-system file.

Smart large objects are also useful for opaque data types with large storage requirements. (See the description of opaque data types in "OPAQUE Data Types" on page 2-61.)

Dynamic Server supports the following smart-large-object data types:

BLOB	Stores binary data. For more information about this data type, see the description on page 2-9.
CLOB	Stores text data. For more information about this data type, see the description on page 2-14.

Dynamic Server stores smart large objects in *sbspaces*. An *sbspace* is a logical storage area that contains one or more chunks that store only BLOB and CLOB data. For information on how to define sbspaces, see your *Performance Guide*.

When you define a BLOB or CLOB column, you can determine the following large-object characteristics:

- LOG and NOLOG: whether the database server should log the smart large object in accordance with the current database log mode
- KEEP ACCESS TIME and NO KEEP ACCESS TIME: whether the database server should keep track of the last time the smart large object was accessed
- HIGH INTEG and MODERATE INTEG: whether the database server should use page headers to detect data corruption

Use of these characteristics can affect performance. For information, see your *Performance Guide*.

When you access a smart-large-object column with an SQL statement, the database server does not send the actual BLOB or CLOB data. Instead, it establishes a pointer to the data and returns this pointer. The client application can then use this pointer to perform the open, read, or write operations on the smart large object.

To access a BLOB or CLOB column from within a client application, use one of the following application programming interfaces (APIs):

- From within an Informix ESQL/C program, use the smart-large-object API.

  For more information, see the *Informix ESQL/C Programmer's Manual*.
- From within a DataBlade module, use the Client and Server API.

  For more information, see the *DataBlade API Programmer's Manual*.

For information on smart large objects, see the *Informix Guide to SQL: Syntax* and *Informix Guide to Database Design and Implementation*.

Time Data Types

You can use DATE, DATETIME, and INTERVAL data in a variety of arithmetic and relational expressions. You can manipulate a DATETIME value with another DATETIME value, an INTERVAL value, the current time (specified by the keyword CURRENT), or some unit of time (identified by the keyword UNITS).

In most situations, you can use a DATE value wherever it is appropriate to use a DATETIME value and vice versa. You also can manipulate an INTERVAL value with the same choices as a DATETIME value. In addition, you can multiply or divide an INTERVAL value by a number.

An INTERVAL column can hold a value that represents the difference between two DATETIME values or the difference between (or sum of) two INTERVAL values. In either case, the result is a span of time, which is an INTERVAL value. Conversely, if you add or subtract an INTERVAL from a DATETIME value, another DATETIME value is produced, because the result is a specific time.

Figure 2-11 lists the binary arithmetic operations that you can perform on DATE, DATETIME, and INTERVAL operands, as well as the data type that is returned by the arithmetic expression.

Figure 2-11
Arithmetic Operations on DATE, DATETIME, and INTERVAL Values

Operand 1	Operator	Operand 2	Result
DATE	-	DATETIME	INTERVAL
DATETIME	-	DATE	INTERVAL
DATE	+ *or* -	INTERVAL	DATETIME
DATETIME	-	DATETIME	INTERVAL
DATETIME	+ *or* -	INTERVAL	DATETIME
INTERVAL	+	DATETIME	DATETIME
INTERVAL	+ *or* -	INTERVAL	INTERVAL
DATETIME	-	CURRENT	INTERVAL
CURRENT	-	DATETIME	INTERVAL
INTERVAL	+	CURRENT	DATETIME
CURRENT	+ *or* -	INTERVAL	DATETIME
DATETIME	+ *or* -	UNITS	DATETIME
INTERVAL	+ *or* -	UNITS	INTERVAL
INTERVAL	* *or* /	NUMBER	INTERVAL

No other combinations are allowed. You cannot add two DATETIME values because this operation does not produce either a specific time or a span of time. For example, you cannot add December 25 and January 1, but you can subtract one from the other to find the time span between them.

Manipulating DATETIME Values

You can subtract most DATETIME values from each other. Dates can be in any order and the result is either a positive or a negative INTERVAL value. The first DATETIME value determines the field precision of the result.

If the second DATETIME value has fewer fields than the first, the second is extended automatically to match the first. (See the discussion of the EXTEND function in the Expression segment in the *Informix Guide to SQL: Syntax*.)

In the following example, subtracting the DATETIME YEAR TO HOUR value from the DATETIME YEAR TO MINUTE value results in a positive interval value of 60 days, 1 hour, and 30 minutes. Because minutes were not included in the second operand, the database server sets the MINUTES value for the second operand to 0 before performing the subtraction.

```
DATETIME (2001-9-30 12:30) YEAR TO MINUTE
   - DATETIME (2001-8-1 11) YEAR TO HOUR

Result: INTERVAL (60 01:30) DAY TO MINUTE
```

If the second DATETIME operand has more fields than the first (regardless of whether the precision of the extra fields is larger or smaller than those in the first operand), the additional time unit fields in the second value are ignored in the calculation.

In the next expression (and its result), the year is not included for the second operand. Therefore, the year is set automatically to the current year (from the system clock-calendar), in this example 2001, and the resulting INTERVAL is negative, which indicates that the second date is later than the first.

```
DATETIME (2001-9-30) YEAR TO DAY
   - DATETIME (10-1) MONTH TO DAY

Result: INTERVAL (1) DAY TO DAY [assuming that the current
                                  year is 2001]
```

Manipulating DATETIME with INTERVAL Values

INTERVAL values can be added to or subtracted from DATETIME values. In either case, the result is a DATETIME value. If you are adding an INTERVAL value to a DATETIME value, the order of values is unimportant; however, if you are subtracting, the DATETIME value must come first. Adding or subtracting a positive INTERVAL value simply moves the DATETIME result forward or backward in time. The expression shown in the following example moves the date ahead by three years and five months:

```
DATETIME (1994-8-1) YEAR TO DAY
   + INTERVAL (3-5) YEAR TO MONTH

Result: DATETIME (1998-01-01) YEAR TO DAY
```

Important: *Evaluate the logic of your addition or subtraction. Remember that months can have 28, 29, 30, or 31 days and that years can have 365 or 366 days.*

In most situations, the database server automatically adjusts the calculation when the initial values do not have the same precision. In certain situations, however, you must explicitly adjust the precision of one value to perform the calculation. If the INTERVAL value you are adding or subtracting has fields that are not included in the DATETIME value, you must use the EXTEND function to explicitly extend the field qualifier of the DATETIME value. (For more information on the EXTEND function, see the Expression segment in the *Informix Guide to SQL: Syntax*.)

For example, you cannot subtract an INTERVAL MINUTE TO MINUTE value from the DATETIME value in the previous example that has a YEAR TO DAY field qualifier. You can, however, use the EXTEND function to perform this calculation, as the following example shows:

```
EXTEND (DATETIME (2001-8-1) YEAR TO DAY, YEAR TO MINUTE)
   - INTERVAL (720) MINUTE(3) TO MINUTE

Result: DATETIME (2001-07-31 12:00) YEAR TO MINUTE
```

The EXTEND function allows you to explicitly increase the DATETIME precision from YEAR TO DAY to YEAR TO MINUTE. This allows the database server to perform the calculation, with the resulting extended precision of YEAR TO MINUTE.

Manipulating DATE with DATETIME and INTERVAL Values

You can use DATE operands in some arithmetic expressions with DATETIME or INTERVAL operands by writing expressions to do the manipulating, as Figure 2-12 shows.

Figure 2-12
Results of Expressions That Manipulate DATE with DATETIME or INTERVAL Values

Expression	Result
DATE – DATETIME	INTERVAL
DATETIME – DATE	INTERVAL
DATE + *or* – INTERVAL	DATETIME

In the cases that Figure 2-12 shows, DATE values are first converted to their corresponding DATETIME equivalents, and then the expression is evaluated by the rules of arithmetic.

Although you can interchange DATE and DATETIME values in many situations, you must indicate whether a value is a DATE or a DATETIME data type. A DATE value can come from the following sources:

- A column or program variable of type DATE
- The TODAY keyword
- The DATE() function
- The MDY function
- A DATE literal

A DATETIME value can come from the following sources:

- A column or program variable of type DATETIME
- The CURRENT keyword
- The EXTEND function
- A DATETIME literal

The database locale defines the default DATE and DATETIME formats. For the default locale, U.S. English, these formats are '*mm* / *dd* / *yy*' for DATE values and '*yyyy*-*mm*-*dd* *hh*:*MM*:*ss*' for DATETIME values.

To represent DATE and DATETIME values as character strings, the fields in the strings must be in proper order. In other words, when a DATE value is expected, the string must be in DATE format and when a DATETIME value is expected, the string must be in DATETIME format. For example, you can use the string `'10/30/2001'` as a DATE string but not as a DATETIME string. Instead, you must use `'2001-10-30'` or `'99-10-30'` as the DATETIME string.

GLS

In a nondefault locale, literal DATE and DATETIME strings must match the formats that the locale defines. For more information, see the *Informix Guide to GLS Functionality*.

You can customize the DATE format that the database server expects with the **DBDATE** and **GL_DATE** environment variables. You can customize the DATETIME format that the database server expects with the **DBTIME** and **GL_DATETIME** environment variables. For more information, see "DBDATE" on page 3-32 and "DBTIME" on page 3-48. For more information on all these environment variables, see the *Informix Guide to GLS Functionality*. ♦

You can also subtract one DATE value from another DATE value, but the result is a positive or negative INTEGER count of days, rather than an INTERVAL value. If an INTERVAL value is required, you can either use the UNITS DAY operator to convert the INTEGER value into an INTERVAL DAY TO DAY value, or else use EXTEND to convert one of the DATE values into a DATETIME value before subtracting.

For example, the following expression uses the DATE( ) function to convert character string constants to DATE values, calculates their difference, and then uses the UNITS DAY keywords to convert the INTEGER result into an INTERVAL value:

```
(DATE ('5/2/1994') - DATE ('4/6/1955')) UNITS DAY

Result: INTERVAL (12810) DAY(5) TO DAY
```

Important: *Because of the high precedence of UNITS relative to other SQL operators, you should generally enclose any arithmetic expression that is the operand of UNITS within parentheses, as in the preceding example.*

If you need YEAR TO MONTH precision, you can use the EXTEND function on the first DATE operand, as the following example shows:

```
EXTEND (DATE ('5/2/1994'), YEAR TO MONTH) - DATE ('4/6/1955')

Result: INTERVAL (39-01) YEAR TO MONTH
```

The resulting INTERVAL precision is YEAR TO MONTH, because the DATETIME value came first. If the DATE value had come first, the resulting INTERVAL precision would have been DAY(5) TO DAY.

Manipulating INTERVAL Values

You can add or subtract INTERVAL values only if both values are from the same class; that is, if both are year-month or both are day-time. In the following example, a SECOND TO FRACTION value is subtracted from a MINUTE TO FRACTION value:

```
INTERVAL (100:30.0005) MINUTE(3) TO FRACTION(4)
   - INTERVAL (120.01) SECOND(3) TO FRACTION

Result: INTERVAL (98:29.9905) MINUTE TO FRACTION(4)
```

The use of numeric qualifiers alerts the database server that the MINUTE and FRACTION in the first value and the SECOND in the second value exceed the default number of digits.

When you add or subtract INTERVAL values, the second value cannot have a field with greater precision than the first. The second INTERVAL, however, can have a field of smaller precision than the first. For example, the second INTERVAL can be HOUR TO SECOND when the first is DAY TO HOUR. The additional fields (in this case MINUTE and SECOND) in the second INTERVAL value are ignored in the calculation.

Multiplying or Dividing INTERVAL Values

You can multiply or divide INTERVAL values by a number. Any remainder from the calculation is ignored, however, and the result is truncated to the precision of the INTERVAL. The following expression multiplies an INTERVAL value by a literal number that has a fractional part:

```
INTERVAL (15:30.0002) MINUTE TO FRACTION(4) * 2.5

Result: INTERVAL (38:45.0005) MINUTE TO FRACTION(4)
```

In this example, $15 * 2.5 = 37.5$ minutes, $30 * 2.5 = 75$ seconds, and $2 * 2.5 = 5$ fraction(4). The 0.5 minute is converted into 30 seconds and 60 seconds are converted into 1 minute, which produces the final result of 38 minutes, 45 seconds, and 0.0005 of a second. The result of any calculation has the same precision as the original INTERVAL operand.

IDS

Extended Data Types

Dynamic Server lets you create extended data types to characterize data that cannot be easily represented with the built-in data types. However, you cannot use extended data types in distributed transactions. You can create the following extended data types:

- Complex data types
- DISTINCT data types
- OPAQUE data types

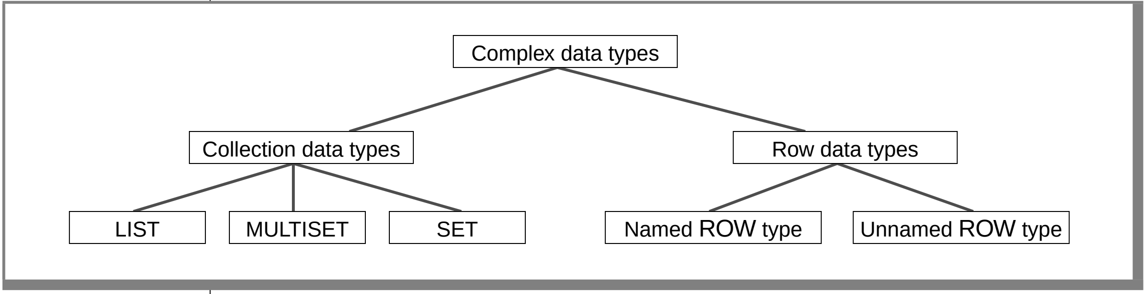
The following sections provide an overview of each of these data types.

For more information about extended data types, see the *Informix Guide to Database Design and Implementation* and *Creating User-Defined Routines and User-Defined Data Types*.

Complex Data Types

A *complex data type* is a data type that you build from other data types (built-in and extended). Figure 2-13 shows the complex types that Dynamic Server supports. The table that follows describes the structure of these data types.

Figure 2-13
Supported Complex Data Types



Data Type	Description
Collection types:	Complex data types that are made up of elements, each of which is of the same data type.
LIST	A group of ordered elements, each of which need not be unique within the group.
MULTISET	A group of elements, each of which need not be unique. The order of the elements is ignored.
SET	A group of elements, each of which is unique. The order of the elements is ignored.
Row types:	Complex data types that are made up of fields.
Named row type	ROW types that are identified by their name.
Unnamed row type	ROW types that are identified by their structure.

Complex data types can be nested. For example, you can construct a row type whose fields include one or more SETs, MULTISETs, row types, and / or LISTs. Likewise, a collection type can have elements whose data type is a row type or a collection type.

All complex types that include opaque data types inherit the following support functions:

input	assign
output	destroy
send	LO handles
recv	hash
import	lessthan
export	equal
import binary	lessthan (row only)
export binary	

The following sections summarize the complex data types. For more information on complex data types, see the *Informix Guide to Database Design and Implementation*.

Collection Data Types

A collection data type is a complex type that is made up of one or more elements. Every element in a collection has the same data type. A collection element can be of any data type (including other complex types) except BYTE, TEXT, SERIAL, or SERIAL8.

Important: *An element cannot have a NULL value. You must specify the NOT NULL constraint for collection elements. No other constraints are valid for collections.*

Dynamic Server supports three kinds of collection types: LIST, SET, and MULTISET. The keywords used to declare these collections are the names of the *type constructors* or just *constructors*. For a description of each of these collection data types, see its entry in this chapter. For the syntax of collection types, see the *Informix Guide to SQL: Syntax*.

Using Complex Data Types in Table Columns

When you specify element values for a collection, list the element values after the constructor and between braces ({ }) symbols. For example, suppose you have a collection column with the following type:

```
CREATE TABLE table1
(
   mset_col MULTISET(INTEGER NOT NULL)
)
```

The following INSERT statement adds one group of element values to this MULTISET column. The word MULTISET in the two examples that follow is the MULTISET constructor.

```
INSERT INTO table1 VALUES (MULTISET{5, 9, 7, 5})
```

Leave the braces empty to indicate an empty set:

```
INSERT INTO table1 VALUE (MULTISET{})
```

An empty collection is not equivalent to a NULL value for the column.

Accessing Collection Data

To access the elements of a collection column, you must fetch the collection into a collection variable and modify the contents of the collection variable. Collection variables can be either of the following types:

- Variables in an SPL routine

  For more information, see the *Informix Guide to SQL: Tutorial*.
- Host variables in an Informix ESQL/C program

  For more information, see the *Informix ESQL/C Programmer's Manual*.

You can also use nested dot notation to access collection data. To learn more about accessing the elements of a collection, see the *Informix Guide to SQL: Tutorial*.

Important: *Collection data types are not valid as arguments to functions that are used for functional indexes.*

Row Data Types

A row type is a sequence of one or more elements, called *fields*. Each field has a name and a data type. The fields of a row are comparable to the columns of a table, but with important differences:

- A field has no DEFAULT clause.
- You cannot define constraints on a field.
- You can only use fields with ROW types, not with tables.

Two kinds of row types exist:

- Named row types are identified by their names.
- Unnamed row types are identified by their structure.

The *structure* of an unnamed row type is the number (and the sequence of data types) of its fields. For more information about row types, see "Row, Named" on page 2-35 and "Row, Unnamed" on page 2-37.

You can cast between named and unnamed row types. For information about casting between row types, see the *Informix Guide to Database Design and Implementation*.

DISTINCT Data Types

A DISTINCT data type has the same internal structure as some other source data type in the database. The source data type can be either a built-in type or an extended type. What distinguishes a DISTINCT type from the source type are the functions defined on this type. For more information, see the description in "Distinct" on page 2-23.

OPAQUE Data Types

An OPAQUE data type is a user-defined data type that is fully encapsulated, that is, whose internal structure is unknown to the database server. For more information, see the description in "Opaque" on page 2-34.

Data Type Casting and Conversion

Occasionally, the data type that was assigned to a column with the CREATE TABLE statement is inappropriate. You might want to change the data type of a column when you need to store larger values than the current data type can accommodate. The database server allows you to change the data type of the column or to cast its values to a different data type with either of the following methods:

- Use the ALTER TABLE statement to modify the data type of a column.

  For example, if you create a SMALLINT column and later find that you need to store integers larger than 32,767, you must change the data type of that column to store the larger value. You can use ALTER TABLE to change the data type to INTEGER. The conversion changes the data type of all values that currently exist in the column as well as any new values that might be added.
- Use the CAST AS keywords or the double colon (::) cast operator to cast a value to a different data type.

  Casting does not permanently alter the data type of a value; it expresses the value in a more convenient form. Casting user-defined data types into built-in types allows client programs to manipulate data types without knowledge of their internal structure.

If you change data types, the new data type must be able to store all the old value

Both data-type conversion and casting depend on casts registered in the **syscasts** system catalog table. For information about **syscasts**, see “SYSCASTS” on page 1-23.

A cast is either built in or user defined. Guidelines exist for casting DISTINCT and extended data types.

For more information about casting opaque types, see *Creating User-Defined Routines and User-Defined Data Types*. For information about casting other extended types, see the *Informix Guide to Database Design and Implementation*.

Using Built-in Casts

User **informix** owns built-in casts. They govern conversions from one built-in data type to another. Built-in casts allow the database server to attempt the following data-type conversions:

- A character type to any other character type
- A character type to or from another built-in type
- A numeric type to any other numeric type

The database server automatically invokes appropriate built-in casts when required. For time data types, conversion between DATE and DATETIME data types requires explicit casts with the EXTEND function, and explicit casts with the UNITS operator are required for number-to-INTERVAL conversion. Built-in casts are not available for converting large (BYTE, BLOB, CLOB, and TEXT) built-in types to other built-in data types.

When you convert a column from one built-in data type to another, the database server applies the appropriate built-in casts to each value already in the column. If the new data type cannot store any of the resulting values, the ALTER TABLE statement fails.

For example, if you try to convert a column from the INTEGER data type to the SMALLINT data type and the following values exist in the INTEGER column, the database server does not change the data type, because SMALLINT columns cannot accommodate numbers greater than 32,767:

```
100    400    700    50000     700
```

The same situation might occur if you attempt to transfer data from FLOAT or SMALLFLOAT columns to INTEGER, SMALLINT, or DECIMAL columns. Errors of overflow, underflow, or truncation can occur during data type conversion.

Sections that follow describe database server behavior during certain types of casts and conversions.

Converting from Number to Number

When you convert data from one number data type to another, you occasionally find rounding errors. The following table indicates which numeric data type conversions are acceptable and what kinds of errors you can encounter when you convert between certain numeric data types.

Target Type →	SMALLINT	INTEGER	INT8	SMALLFLOAT	FLOAT	DECIMAL
SMALLINT	OK	OK	OK	OK	OK	OK
INTEGER	E	OK	OK	E	OK	P
INT8	E	E	OK	D	E	P
SMALLFLOAT	E	E	E	OK	OK	P
FLOAT	E	E	E	D	OK	P
DECIMAL	E	E	E	D	D	P

Legend:
OK = No error
P = An error can occur depending on the precision of the DECIMAL
E = An error can occur depending on the data value
D= No error but less significant digits might be lost

For example, if you convert a FLOAT value to DECIMAL(4,2), your database server rounds off the floating-point numbers before storing them as decimal numbers. This conversion can result in an error depending on the precision assigned to the DECIMAL column.

Converting Between Number and CHAR

You can convert a CHAR (or NCHAR) column to a numeric column. However, if the CHAR or NCHAR column contains any characters that are not valid in a number column (for example, the letter *l* instead of the number *1*), your database server returns an error.

You can also convert a numeric column to a character column. However, if the character column is not large enough to receive the number, the database server generates an error.

If the database server generates an error, it cannot complete the ALTER TABLE statement or cast and leaves the column values as characters. You receive an error message and the statement is rolled back (whether you are in a transaction or not).

Converting Between INTEGER and DATE or DATETIME

You can convert an integer column (SMALLINT, INTEGER, or INT8) to a DATE or DATETIME value. The database server interprets the integer as a value in the internal format of the DATE or DATETIME column. You can also convert a DATE or DATETIME column to an integer column. The database server stores the internal format of the DATE or DATETIME column as an integer.

For a DATE column, the internal format is a Julian date. For a DATETIME column, the internal format stores the date and time in a condensed integer format.

Converting Between DATE and DATETIME

You can convert DATE columns to DATETIME columns. However, if the DATETIME column contains more fields than the DATE column, the database server either ignores the fields or fills them with zeros. The illustrations in the following list show how these two data types are converted (assuming that the default date format is *mm*/*dd*/*yyyy*):

- If you convert DATE to DATETIME YEAR TO DAY, the database server converts the existing DATE values to DATETIME values. For example, the value 08/15/2002 becomes 2002-08-15.
- If you convert DATETIME YEAR TO DAY to the DATE format, the value 2002-08-15 becomes 08/15/2002.
- If you convert DATE to DATETIME YEAR TO SECOND, the database server converts existing DATE values to DATETIME values and fills in the additional DATETIME fields with zeros. For example, 08/15/2002 becomes 2002-08-15 00:00:00.
- If you convert DATETIME YEAR TO SECOND to DATE, the database server converts existing DATETIME to DATE values but drops fields more precise than DAY. For example, 2002-08-15 12:15:37 becomes 08/15/2002.

Using User-Defined Casts

Implicit and explicit casts are owned by the users who create them. They govern casts and conversions between user-defined data types and other data types.

Developers of user-defined data types must create certain implicit and explicit casts and the functions that are used to implement them. The casts allow user-defined types to be expressed in a form that clients can manipulate.

For information on how to register and use implicit and explicit casts, see the CREATE CAST statement in the *Informix Guide to SQL: Syntax* and the *Informix Guide to Database Design and Implementation*.

Implicit Casts

The database server automatically invokes a single implicit cast when needed to evaluate and compare expressions or pass arguments. Operations that require more than one implicit cast fail.

Implicit casts allow you to convert a user-defined data type to a built-in type or vice versa.

Users can explicitly invoke an implicit cast using the CAST AS keywords or the double colon (**::**) cast operator.

Explicit Casts

Explicit casts, unlike implicit casts or built-in casts, are *never* invoked automatically by the database server. Users must invoke them explicitly with the CAST AS keywords or with the double colon (**::**) cast operator.

Explicit casts do not allow you to convert a user-defined data type to a built-in data type or vice versa.

Determining Which Cast to Apply

The database server uses the following rules to determine which cast to apply in a particular situation:

- To compare two built-in types, the database server automatically invokes the appropriate built-in casts.
- The database server applies only one implicit cast per operand. If two or more casts are needed to convert the operand to the desired type, the user must explicitly invoke the additional casts.

  In the following example, the literal value `5.55` is implicitly cast to DECIMAL, and is then explicitly cast to MONEY, and finally to yen:

  ```
  CREATE DISTINCT TYPE yen AS MONEY
  . . .
  INSERT INTO currency_tab
     VALUES (5.55::MONEY::yen)
  ```
- To compare a DISTINCT type to its source type, the user must explicitly cast one type to the other.
- To compare a DISTINCT type to a type other than its source, the database server looks for an implicit cast between the source type and the desired type.

  If neither cast is registered, the user must invoke an explicit cast between the DISTINCT type and the desired type. If this cast is not registered, the database server automatically invokes a cast from the source type to the desired type.

  If none of these casts is defined, the comparison fails.
- To compare an OPAQUE type to a built-in type, the user must explicitly cast the opaque type to a data type that the database server understands (such as LVARCHAR, SENDRECV, IMPEX, or IMPEXBIN). The database server then invokes built-in casts to convert the results to the desired built-in type.
- To compare two opaque types, the user must explicitly cast one opaque type to a form that the database server understands (such as LVARCHAR, SENDRECV, IMPEX, or IMPEXBIN), and then explicitly cast this type to the second opaque type.

For information about casting and the IMPEX, IMPEXBIN, LVARCHAR, and SENDRECV types, see *Creating User-Defined Routines and User-Defined Data Types*.

Casts for Distinct Types

You define a distinct type based on a built-in type or an existing opaque type or row type. Although data of the distinct type has the same length and alignment and is passed in the same way as data of the source type, the two cannot be compared directly. To compare a distinct type and its source type, you must explicitly cast one type to the other.

When you create a new distinct type, the database server automatically registers two explicit casts:

- A cast from the distinct type to its source type
- A cast from the source type to the distinct type

You can create an implicit cast between a distinct type and its source type. To create an implicit cast, however, you must first drop the default explicit cast between the distinct type and its source type.

You also can use all casts that have been registered for the source type without modification on the distinct type. You can also create and register new casts and support functions that apply *only* to the distinct type.

For examples that show how to create a cast function for a distinct type and register the function as cast, see the *Informix Guide to Database Design and Implementation*.

Important: *For releases of Dynamic Server earlier than Version 9.21, distinct data types inherited the built-in casts that are provided for the source type. The built-in casts of the source type are not inherited by distinct data types in this release.*

What Extended Data Types Can Be Cast?

The following table shows the extended data type combinations that you can cast. The table shows only whether or not a cast between a source type and a target type are possible. In some cases, you must first create a user-defined cast before you can perform a conversion between two data types. In other cases, the database server provides either an implicit cast or a built-in cast that you must explicitly invoke.

Target Type →	Opaque Type	Distinct Type	Named Row Type	Unnamed Row Type	Collection Type	Built-in Type
Opaque Type	Explicit or implicit	Explicit	Explicit	Not Allowed	Not Allowed	Explicit or implicit[3]
Distinct Type	Explicit[3]	Explicit	Explicit	Not Allowed	Not Allowed	Explicit or implicit
Named Row Type	Explicit[3]	Explicit	Explicit[3]	Explicit[1]	Not Allowed	Not Allowed
Unnamed Row Type	Not Allowed	Not Allowed	Explicit[1]	Implicit[1]	Not Allowed	Not Allowed
Collection Type	Not Allowed	Not Allowed	Not Allowed	Not Allowed	Explicit[2]	Not Allowed
Built-in Type	Explicit or implicit[3]	Explicit or implicit	Not Allowed	Not Allowed	Not Allowed	System defined (implicit)

[1] Applies when two row types are structurally equivalent or casts exist to handle data conversions where corresponding field types are not the same.

[2] Applies when a cast exists to convert between the element types of the respective collection types.

[3] Applies when a user-defined cast exists to convert between the two data types.

Operator Precedence

An *operator* is a symbol or keyword that can appear in an SQL expression. Most SQL operators are restricted in the data types of their operands and returned values. Some operators only support operands of built-in data types; others can support built-in and extended data types as operands.

The following table shows the precedence of the operators that Informix database servers support, in descending (highest to lowest) order of precedence. Operators with the same precedence are listed in the same row.

Operator Precedence	Example in Expression
. (*membership*) **[ ]** (*substring*)	**customer.phone [**1, 3**]**
UNITS	**d** **x** UNITS DAY
+ **-** (*unary*)	**- y**
:: (*cast*)	NULL**::**TEXT
***** **/**	**x / y**
+ **-** (*binary*)	**x -y**
|| (*concatenation*)	**customer.fname || customer.lname**
ANY ALL SOME	**orders.ship_date** > SOME (SELECT **paid_date** FROM **orders**)
NOT	NOT **y**
< **<=** **=** **>** **>=** **!=** **<>** IN BETWEEN LIKE MATCHES	**customer.fname** MATCHES **y**
AND	**x** AND **y**
OR	**x** OR **y**

See the *Informix Guide to SQL: Syntax* for the syntax and semantics of these SQL operators.

Chapter

Environment Variables

3

In This Chapter 3-5

Types of Environment Variables 3-5

Where to Set Environment Variables on UNIX 3-6

Where to Set Environment Variables on Windows 3-7

Using Environment Variables on UNIX 3-8
- Setting Environment Variables in a Configuration File 3-8
- Setting Environment Variables at Login Time 3-9
- Syntax for Setting Environment Variables. 3-9
- Unsetting Environment Variables 3-10
- Modifying an Environment-Variable Setting. 3-10
- Viewing Your Environment-Variable Settings 3-11
- Checking Environment Variables with the chkenv Utility 3-11
- Rules of Precedence 3-12

Using Environment Variables on Windows 3-13
- Setting Environment Variables for Native Windows Applications . 3-13
- Setting Environment Variables for Command-Prompt Utilities . . 3-15
  - Using the System Applet to Work with Environment Variables . 3-15
  - Using the Command Prompt to Work with Environment Variables 3-16
  - Using dbservername.cmd to Initialize a Command-Prompt Environment 3-17
- Rules of Precedence 3-18

List of Environment Variables 3-18

Environment Variables. 3-23
AC_CONFIG. 3-24
AFDEBUG 3-24
CPFIRST 3-25
DBACCNOIGN. 3-26
DBANSIWARN 3-27
DBBLOBBUF. 3-28
DBCENTURY 3-28
DBDATE 3-32
DBDELIMITER 3-35
DBEDIT 3-35
DBFLTMASK 3-36
DBLANG 3-37
DBMONEY 3-38
DBNLS. 3-40
DBONPLOAD 3-41
DBPATH 3-42
DBPRINT 3-44
DBREMOTECMD 3-45
DBSPACETEMP 3-46
DBTEMP 3-48
DBTIME 3-48
DBUPSPACE. 3-52
DEFAULT_ATTACH 3-53
DELIMIDENT 3-53
ENVIGNORE 3-54
FET_BUF_SIZE 3-55
IFMX_SMLTBL_BROADCAST_SIZE 3-56
IFX_DEF_TABLE_LOCKMODE 3-56
IFX_DIRECTIVES 3-58
IFX_LONGID 3-59
IFX_NETBUF_PVTPOOL_SIZE 3-60
IFX_NETBUF_SIZE 3-60
IFX_UPDDESC 3-61
IMCADMIN 3-61
IMCCONFIG 3-62

IMCSERVER 3-62
INFORMIXC 3-63
INFORMIXCONCSMCFG 3-63
INFORMIXCONRETRY 3-64
INFORMIXCONTIME 3-65
INFORMIXCPPMAP 3-66
INFORMIXDIR 3-67
INFORMIXKEYTAB. 3-67
INFORMIXOPCACHE 3-68
INFORMIXSERVER 3-68
INFORMIXSHMBASE 3-69
INFORMIXSQLHOSTS. 3-70
INFORMIXSTACKSIZE 3-71
INFORMIXTERM 3-72
INF_ROLE_SEP 3-73
INTERACTIVE_DESKTOP_OFF 3-74
ISM_COMPRESSION 3-74
ISM_DEBUG_FILE 3-75
ISM_DEBUG_LEVEL 3-75
ISM_ENCRYPTION 3-76
ISM_MAXLOGSIZE. 3-76
ISM_MAXLOGVERS 3-77
JAR_TEMP_PATH 3-77
JAVA_COMPILER 3-78
JVM_MAX_HEAP_SIZE 3-78
LD_LIBRARY_PATH 3-79
LIBPATH 3-79
NODEFDAC 3-80
ONCONFIG 3-80
OPTCOMPIND 3-81
OPTMSG 3-82
OPTOFC. 3-83
OPT_GOAL 3-83
PATH 3-84
PDQPRIORITY 3-85
PLCONFIG. 3-87

PLOAD_LO_PATH 3-87
PLOAD_SHMBASE 3-88
PSORT_DBTEMP 3-89
PSORT_NPROCS 3-89
SHLIB_PATH 3-91
STMT_CACHE 3-92
TERM 3-92
TERMCAP 3-93
TERMINFO 3-94
THREADLIB. 3-94
XFER_CONFIG 3-95

Index of Environment Variables 3-95

In This Chapter

Various *environment variables* affect the functionality of your Informix products. You can set environment variables that identify your terminal, specify the location of your software, and define other parameters.

Some environment variables are required; others are optional. You must either set or accept the default setting for required environment variables.

This chapter describes how to use the environment variables that apply to one or more Informix products and shows how to set them.

Types of Environment Variables

The following types of environment variables are discussed in this chapter:

- Informix-specific environment variables

  Set Informix environment variables when you want to work with Informix products. Each Informix product manual specifies the environment variables that you must set to use that product.

- Operating-system-specific environment variables

  Informix products rely on the correct setting of certain standard operating-system environment variables. For example, you must always set the **PATH** environment variable.

In a UNIX environment, you might also have to set the **TERMCAP** or **TERMINFO** environment variable to use some products effectively.

GLS

The GLS environment variables that support nondefault locales are described in the *Informix Guide to GLS Functionality*. These GLS variables are included in the list of environment variables in Figure 3-1 on page 3-19 and in the topic index in Figure 3-2 on page 3-96, but are not discussed in this manual. ♦

***Tip:** Additional environment variables that are specific to your client application or SQL API might be discussed in the manual for that product.*

UNIX

Where to Set Environment Variables on UNIX

You can set environment variables on UNIX in the following places:

- At the system prompt on the command line

  When you set an environment variable at the system prompt, you must reassign it the next time you log into the system. For more information, see "Using Environment Variables on UNIX" on page 3-8.

- In an environment-configuration file

  An environment-configuration file is a common or private file where you can set all the environment variables that Informix products use. The use of such files reduces the number of environment variables that you must set at the command line or in a shell file.

- In a login file

  Values of environment variables set in your **.login**, **.cshrc**, or **.profile** file are assigned automatically every time you log into the system.

E/C

In Informix ESQL/C, you can set supported environment variables within an application with the **putenv( )** system call and retrieve values with the **getenv( )** system call, if your UNIX system supports these functions. For more information on **putenv( )** and **getenv( )**, see the *Informix ESQL/C Programmer's Manual* and your C documentation. ♦

WIN NT

Where to Set Environment Variables on Windows

You might be able to set environment variables in several places on Windows, depending on which Informix application you use.

For native Windows Informix applications, such as the database server, environment variables can be set only in the Windows registry. Environment variables set in the registry cannot be modified elsewhere.

For utilities that run in a command-prompt session, such as **dbaccess**, environment variables can be set in several ways, as described in "Setting Environment Variables for Command-Prompt Utilities" on page 3-15.

To use client applications such as ESQL/C or the Schema Tools on Windows environment, use the Setnet32 utility to set environment variables. For information about the Setnet32 utility, see the *Informix Client Products Installation Guide* for your operating system.

E/C

In Informix ESQL/C, you can set supported environment variables within an application with the **ifx_putenv( )** function and retrieve values with the **ifx_getenv( )** function, if your Windows system supports them. For more information on **ifx_putenv( )** and **ifx_getenv( )**, see the *Informix ESQL/C Programmer's Manual*. ♦

UNIX

Using Environment Variables on UNIX

The following sections discuss setting, unsetting, modifying, and viewing environment variables. If you already use an Informix product, some or all of the appropriate environment variables might be set.

Setting Environment Variables in a Configuration File

The common (shared) environment-configuration file that is provided with Informix products resides in **$INFORMIXDIR/etc/informix.rc**. Permissions for this shared file must be set to 644.

A user can override the system or shared environment variables by setting variables in a private environment-configuration file. This file must have all of the following characteristics:

- Stored in the user's home directory
- Named **.informix**
- Permissions set to readable by the user

An environment-configuration file can contain comment lines (preceded by the # comment indicator) and variable definition lines that set values (separated by blank spaces or tabs), as the following example shows:

```
# This is an example of an environment-configuration file
#
DBDATE DMY4-
#
# These are ESQL/C environment variable settings
#
INFORMIXC gcc
CPFIRST TRUE
```

You can use the **ENVIGNORE** environment variable, described in "ENVIGNORE" on page 3-54, to override one or more entries in an environment-configuration file. Use the Informix **chkenv** utility, described in "Checking Environment Variables with the chkenv Utility" on page 3-11, to perform a sanity check on the contents of an environment-configuration file. The **chkenv** utility returns an error message if the file contains a bad environment variable or if the file is too large.

The first time you set an environment variable in a shell file or environment-configuration file, you must tell the shell process to read your entry before you work with your Informix product. If you use a C shell, **source** the file; if you use a Bourne or Korn shell, use a period (**.**) to execute the file.

Setting Environment Variables at Login Time

Add the commands that set your environment variables to the appropriate login file:

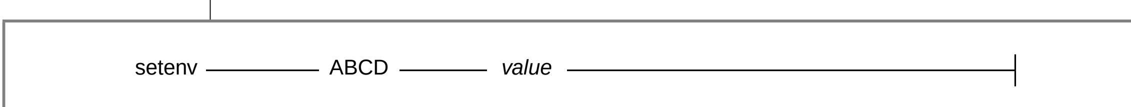
For the C shell	**.login** or **.cshrc**
For the Bourne shell or Korn shell	**.profile**

Syntax for Setting Environment Variables

Use standard UNIX commands to set environment variables. The examples in the following table show how to set the ABCD environment variable to *value* for the C shell, Bourne shell, and Korn shell. The Korn shell also supports a shortcut, as the last row indicates. Environment variables are case sensitive.

Shell	Command
C	`setenv ABCD value`
Bourne	`ABCD=value` `export ABCD`
Korn	`ABCD=value` `export ABCD`
Korn	`export ABCD=value`

The following diagram shows how the syntax for setting an environment variable is represented throughout this chapter. These diagrams indicate the setting for the C shell; for the Bourne or Korn shells, use the syntax illustrated in the preceding table.



For more information on how to read syntax diagrams, see "Syntax Conventions" on page 10 of the Introduction.

Unsetting Environment Variables

To unset an environment variable, enter the following command.

Shell	Command
C	`unsetenv ABCD`
Bourne or Korn	`unset ABCD`

Modifying an Environment-Variable Setting

Sometimes you must add information to an environment variable that is already set. For example, the **PATH** environment variable is always set on UNIX. When you use an Informix product, you must add to the **PATH** setting the name of the directory where the executable files for the Informix products are stored.

In the following example, the **INFORMIXDIR** is **/usr/informix**. (That is, during installation, the Informix products were installed in the **/usr /informix** directory.) The executable files are in the **bin** subdirectory, **/usr/informix/bin**. To add this directory to the front of the C shell **PATH** environment variable, use the following command:

```
setenv PATH /usr/informix/bin:$PATH
```

Rather than entering an explicit pathname, you can use the value of the **INFORMIXDIR** environment variable (represented as **$**INFORMIXDIR), as the following example shows:

```
setenv INFORMIXDIR /usr/informix
setenv PATH $INFORMIXDIR/bin:$PATH
```

You might prefer to use this version to ensure that your **PATH** entry does not conflict with the search path that was set in **INFORMIXDIR,** and so that you do not have to reset **PATH** whenever you change **INFORMIXDIR**.

If you set the **PATH** environment variable on the C shell command line, you might need to include braces ({ }) with the existing **INFORMIXDIR** and **PATH**, as the following command shows:

```
setenv PATH ${INFORMIXDIR}/bin:${PATH}
```

For more information about how to set and modify environment variables, refer to the manuals for your operating system.

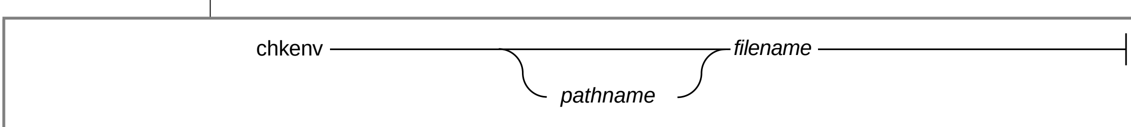
Viewing Your Environment-Variable Settings

After you have installed one or more Informix products, enter the following command at the system prompt to view your current environment settings.

UNIX Version	Command
BSD UNIX	`env`
UNIX System V	`printenv`

Checking Environment Variables with the chkenv Utility

The **chkenv** utility checks the validity of shared or private environment-configuration files. It validates the names of the environment variables in the file, but not their values. Use **chkenv** to provide debugging information when you define, in an environment-configuration file, all the environment variables that your Informix products use.



filename is the name of the environment-configuration file to be debugged.

pathname is the full directory path in which the environment variable file is located.

File **$INFORMIXDIR/etc/informix.rc** is the shared environment-configuration file. A private environment-configuration file is stored as **.informix** in the home directory of the user. If you specify no *pathname* for **chkenv**, the utility checks both the shared and private environment configuration files. If you provide a pathname, **chkenv** checks only the specified file.

Issue the following command to check the contents of the shared environment-configuration file:

```
chkenv informix.rc
```

The **chkenv** utility returns an error message if it finds a bad environment-variable name in the file or if the file is too large. You can modify the file and rerun the utility to check the modified environment-variable names.

Informix products ignore all lines in the environment-configuration file, starting at the point of the error, if the **chkenv** utility returns the following message:

```
-33523  filename: Bad environment variable on line number.
```

If you want the product to ignore specified environment-variables in the file, you can also set the **ENVIGNORE** environment variable. For a discussion of the use and format of environment-configuration files and the **ENVIGNORE** environment variable, see page 3-54.

Rules of Precedence

When an Informix product accesses an environment variable, normally the following rules of precedence apply:

1. Of highest precedence is the value that is defined in the environment (shell) by explicitly setting the value at the shell prompt.
2. The second highest precedence goes to the value that is defined in the private environment-configuration file in the home directory of the user (**~/.informix**).
3. The next highest precedence goes to the value that is defined in the common environment-configuration file ($**INFORMIXDIR/etc/informix.rc**).
4. The lowest precedence goes to the default value, if one exists.

For precedence information about GLS environment variables, see the *Informix Guide to GLS Functionality*.

***Important:** If you set one or more environment variables before you start the database server, and you do not explicitly set the same environment variables for your client products, the clients will adopt the original settings.*

WIN NT

Using Environment Variables on Windows

The following sections discuss setting, viewing, unsetting, and modifying environment variables for native Windows applications and command-prompt utilities.

Setting Environment Variables for Native Windows Applications

Native Windows Informix applications, such as the database server itself, store their configuration information in the Windows registry. To modify this information, you must use the Registry Editor, **regedt32.exe**.

To manipulate environment variables with the Registry Editor

1. Launch the Registry Editor, **regedt32.exe**, and choose the window titled **HKEY_LOCAL_MACHINE**.
2. In the left pane, double-click the SOFTWARE registry key (shown as a small, yellow file folder icon).

   The SOFTWARE registry key expands to show several subkeys, one of which is Informix. Continue down the tree in the following sequence:

   `OnLine,` *`dbservername`*`, Environment.`

   Substitute the name of your database server in place of *dbservername*.
3. With the Environment registry key selected in the left pane, you should see a list of environment variables and their values in the right pane (for example, **CLIENT_LOCALE:REG_SZ:EN_US.CP1252**).

4. Change existing environment variables if needed.
   a. Double-click the environment variable.
   b. Type the new value in the String Editor dialog box.
   c. Click OK to accept the value.
5. Add new environment variables if needed.
   a. Choose **Edit→Add Value** in the Registry Editor.
   b. Enter the name of the environment variable in the Value Name edit box and choose REG_SZ as the data type.
   c. Click OK and type a value for the environment variable in the String Editor dialog box.
6. Delete an environment variable, if needed.
   a. Select the variable name.
   b. Choose **Edit→Delete** in the Registry Editor.

For more information on how to use the Registry Editor, see your operating-system documentation.

Important: *In order to use the Registry Editor to change database server environment variables, you must belong to either the Administrators' or Informix-Admin groups. For information on assigning users to groups, see your operating-system documentation.*

Setting Environment Variables for Command-Prompt Utilities

You can set environment variables for command-prompt utilities in the following ways:

- With the System applet in the Control Panel
- In a command-line session

Using the System Applet to Work with Environment Variables

The System applet provides a graphical interface to create, modify, and delete system-wide and user-specific variables. Environment variables that are set with the System applet are visible to all command-prompt sessions.

To change environment variables with the System applet in the control panel

1. Double-click the System applet icon from the Control Panel window.

   Click the Environment tab near the top of the window. Two list boxes display System Environment Variables and User Environment Variables. System Environment Variables apply to an entire system, and User Environment Variables apply only to the sessions of the individual user.
2. To change the value of an existing variable, select that variable.

   The name of the variable and its current value appear in the boxes at the bottom of the window.
3. Highlight the existing value and type the new value.
4. To add a new variable, highlight an existing variable and type the new variable name in the box at the bottom of the window.
5. Next, enter the value for the new variable at the bottom of the window and click the **Set** button.
6. To delete a variable, select the variable and click the **Delete** button.

Important: *In order to use the System applet to change System environment variables, you must belong to the Administrators' group. For information on assigning users to groups, see your operating-system documentation.*

Using the Command Prompt to Work with Environment Variables

The following diagram shows the syntax for setting an environment variable at a command prompt in Windows.

```
set ——— ABCD ——— = ——— value ———————————|
```

If no *value* is specified, the environment variable is unset, as if it did not exist.

For more information on how to read syntax diagrams, see "Syntax Conventions" on page 10 of the introduction.

To view your current settings after one or more Informix products are installed, enter the following command at the command prompt.

```
set ————————————————————————————————————|
```

Sometimes you must add information to an environment variable that is already set. For example, the **PATH** environment variable is always set in Windows environments. When you use an Informix product, you must add the name of the directory where the executable files for the Informix products are stored to the **PATH**.

In the following example, **INFORMIXDIR** is **d:\informix**, (that is, during installation, Informix products were installed in the **d: \informix** directory). The executable files are in the **bin** subdirectory, **d:\informix\bin**. To add this directory at the beginning of the **PATH** environment-variable value, use the following command:

```
set PATH=d:\informix\bin;%PATH%
```

Rather than entering an explicit pathname, you can use the value of the **INFORMIXDIR** environment variable (represented as **%INFORMIXDIR%**), as the following example shows:

```
set INFORMIXDIR=d:\informix
set PATH=%INFORMIXDIR%\bin;%PATH%
```

You might prefer to use this version to ensure that your **PATH** entry does not contradict the search path that was set in **INFORMIXDIR**, and to avoid the need to reset **PATH** whenever you change **INFORMIXDIR**.

For more information about setting and modifying environment variables, refer to your operating-system manuals.

Using dbservername.cmd to Initialize a Command-Prompt Environment

Each time that you open a Windows command prompt, it acts as an independent environment. Therefore, environment variables that you set within it are valid only for that particular command-prompt instance.

For example, if you open one command window and set the variable, **INFORMIXDIR**, and then open another command window and type `set` to check your environment, you will find that **INFORMIXDIR** is not set in the new command-prompt session.

The database server installation program creates a *batch file* that you can use to configure command-prompt utilities, ensuring that your command-prompt environment is initialized correctly each time that you run a command-prompt session. The batch file, ***dbservername*.cmd**, is located in **%INFORMIXDIR%**, and is a plain text file that you can modify with any text editor. If you have more than one database server installed in **%INFORMIXDIR%**, there will be more than one batch file with the **.cmd** extension, each bearing the name of the database server with which it is associated.

To run ***dbservername*.cmd** from a command prompt, type ***dbservername*** or configure a command prompt so that it runs ***dbservername*.cmd** automatically at start up.

Rules of Precedence

When an Informix product accesses an environment variable, normally the following rules of precedence apply:

1. The highest precedence goes to the value that is defined in the environment by explicitly setting the value at the command prompt.
2. The second highest precedence goes to the value that is defined in the System control panel as a User Environment Variable.
3. The third highest precedence goes to the value that is defined in the System control panel as a System Environment Variable.
4. The lowest precedence goes to the default value.

Important: *Because Windows services access only environment variables that are set in the registry, the preceding rules of precedence do not apply for Informix native Windows applications. For native Windows applications, the highest precedence goes to variables that are explicitly defined in the registry, and the lowest precedence goes to the default value. In addition, if you set one or more environment variables before you start the database server, and you do not explicitly set the same environment variables for client products, the clients will adopt the original settings.*

List of Environment Variables

Figure 3-1 on page 3-19 contains an alphabetical list of the environment variables that you can set for an Informix database server and SQL API products. Most of these environment variables are described in this chapter on the pages listed in the "Page" column. The ✔ symbol indicates that XPS or Dynamic Server (or both, if both columns are so marked) support the environment variable.

GLS

The notation "GLS" in the "Page" column indicates a GLS environment variables that is valid in nondefault locales, and that is described in the *Informix Guide to GLS Functionality*. ♦

Figure 3-1
Alphabetical List Of Environment Variables

Environment Variable	XPS	IDS	Restrictions	Page
AC_CONFIG	✔	✔	None	3-24
AFDEBUG				3-24
CC8BITLEVEL			ESQL/C only	GLS
CLIENT_LOCALE	✔	✔	None	GLS
CPFIRST	✔	✔	None	3-25
DBACCNOIGN	✔	✔	DB-Access only	3-26
DBANSIWARN	✔	✔	None	3-27
DBBLOBBUF	✔	✔	None	3-28
DBCENTURY			SQL APIs only	3-28
DBDATE	✔	✔	None	3-32; GLS
DBDELIMITER	✔	✔	None	3-35
DBEDIT	✔	✔	None	3-35
DBFLTMASK	✔	✔	DB-Access only	3-36
DBLANG	✔	✔	None	3-37; GLS
DBMONEY	✔	✔	None	3-38; GLS
DBNLS		✔		3-40
DBONPLOAD		✔	HPL only	3-41
DBPATH	✔	✔	None	3-42
DBPRINT	✔	✔	UNIX only	3-44
DBREMOTECMD	✔	✔	UNIX only	3-45
DBSPACETEMP	✔	✔	None	3-46
DBTEMP			DB-Access, Gateways only	3-48

(1 of 5)

Environment Variable	XPS	IDS	Restrictions	Page
DBTIME			SQL APIs only	3-48; GLS
DBUPSPACE	✔	✔	None	3-52
DB_LOCALE	✔	✔	None	GLS
DEFAULT_ATTACH		✔	Deprecated	3-53
DELIMIDENT	✔	✔	None	3-53
ENVIGNORE	✔	✔	UNIX only	3-54
ESQLMF	✔	✔	ESQL/C only	GLS
FET_BUF_SIZE	✔	✔	SQL APIs, DB-Access only	3-55
GLS8BITSYS	✔	✔	None	GLS
GL_DATE	✔	✔	None	GLS
GL_DATETIME	✔	✔	None	GLS
IFMX_SMLTBL_BROADCAST_SIZE	✔		None	3-56
IFX_DEF_TABLE_LOCKMODE	✔	✔	None	3-56
IFX_DIRECTIVES	✔	✔	None	3-58
IFX_LONGID		✔	None	3-59
IFX_NETBUF_PVTPOOL_SIZE	✔	✔	UNIX only	3-60
IFX_NETBUF_SIZE	✔	✔	None	3-60
IFX_UPDDESC		✔	None	3-61
IMCADMIN		✔		3-61
IMCCONFIG		✔		3-62
IMCSERVER		✔		3-62
INFORMIXC			ESQL/C, UNIX only	3-63
INFORMIXCONCSMCFG		✔	None	3-63

(2 of 5)

Environment Variable	XPS	IDS	Restrictions	Page
INFORMIXCONRETRY	✔	✔	None	3-64
INFORMIXCONTIME	✔	✔	None	3-65
INFORMIXCPPMAP		✔	None	3-66
INFORMIXDIR	✔	✔	None	3-67
INFORMIXKEYTAB	✔	✔	UNIX only	3-67
INFORMIXOPCACHE		✔	Optical Subsystem only	3-68
INFORMIXSERVER	✔	✔	None	3-68
INFORMIXSHMBASE	✔	✔	UNIX only	3-69
INFORMIXSQLHOSTS	✔	✔	None	3-70
INFORMIXSTACKSIZE	✔	✔	None	3-71
INFORMIXTERM	✔	✔	DB-Access, UNIX only	3-72
INF_ROLE_SEP		✔	None	3-73
INTERACTIVE_DESKTOP_OFF		✔	NT only	3-74
ISM_COMPRESSION	✔	✔	ISM, ON-Bar only	3-74
ISM_DEBUG_FILE	✔	✔	ISM only	3-75
ISM_DEBUG_LEVEL	✔	✔	ISM, ON-Bar only	3-75
ISM_ENCRYPTION	✔	✔	ISM, ON-Bar only	3-76
ISM_MAXLOGSIZE	✔	✔	ISM only	3-76
ISM_MAXLOGVERS	✔	✔	ISM only	3-77
JAR_TEMP_PATH		✔		3-77
JAVA_COMPILER		✔		3-78
JVM_MAX_HEAP_SIZE		✔		3-78

(3 of 5)

Environment Variable	XPS	IDS	Restrictions	Page
LD_LIBRARY_PATH			SQL APIs, UNIX only	3-79
LIBPATH			SQL APIs, UNIX only	3-79
NODEFDAC	✔	✔	None	3-80
ONCONFIG	✔	✔	None	3-80
OPTCOMPIND	✔	✔	None	3-81
OPTMSG			ESQL/C only	3-82
OPTOFC			ESQL/C only	3-83
OPT_GOAL	✔	✔	UNIX only	3-83
PATH	✔	✔	None	3-84
PDQPRIORITY	✔	✔	None	3-85
PLCONFIG		✔	HPL only	3-87
PLOAD_LO_PATH		✔	HPL only	3-87
PLOAD_SHMBASE		✔	HPL only	3-88
PSORT_DBTEMP	✔	✔	None	3-89
PSORT_NPROCS	✔	✔	None	3-89
SERVER_LOCALE	✔	✔	None	GLS
SHLIB_PATH	✔	✔	UNIX only	3-91
STMT_CACHE		✔	None	3-92
TERM	✔	✔	UNIX only	3-92
TERMCAP	✔	✔	UNIX only	3-93

(4 of 5)

Environment Variable	XPS	IDS	Restrictions	Page
TERMINFO	✔	✔	UNIX only	3-94
THREADLIB			ESQL/C, UNIX only	3-94
XFER_CONFIG	✔		None	3-95

(5 of 5)

Tip: *You might encounter references to environment variables that are not listed in Figure 3-1. Most likely, these environment variables are not supported in this release or are used to maintain backward compatibility with certain earlier product versions. For information, refer to an earlier version of your Informix documentation.*

Environment Variables

Sections that follow discuss environment variables that Informix products use.

Important: *The descriptions of the following environment variables include the syntax for setting the environment variable on* UNIX. *For a general description of how to set these environment variables on Windows, see "Setting Environment Variables for Native Windows Applications" on page 3-13 and "Setting Environment Variables for Command-Prompt Utilities" on page 3-15.*

AC_CONFIG

You can set the **AC_CONFIG** environment variable to specify the path for the **ac_config.std** configuration file for the **archecker** utility. The **archecker** utility checks the validity and completeness of an ON-Bar storage-space backup. The **ac_config.std** file contains default **archecker** configuration parameters.

```
setenv ———————— AC_CONFIG ———— pathname ————┤
```

pathname is the location of the **ac_config.std** configuration file in **$INFORMIXDIR/etc** or **%INFORMIXDIR%\etc**.

For information on **archecker**, see your *Backup and Restore Guide*.

AFDEBUG

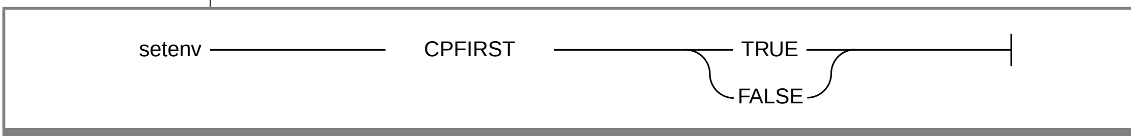
You can create files to hold verbose messages from the Java virtual machine (JVM) about releasing memory that had been allocated to objects by setting the **AFDEBUG** environment variable.

```
setenv ———————— AFDEBUG ————————————┤
```

No value is required. You can also set the configuration parameter AFCRASH to 0x00000010 to achieve the same result.

CPFIRST

Set the **CPFIRST** environment variable to specify the default compilation order for all ESQL/C source files in your programming environment.



When compiling an ESQL/C program with **CPFIRST** not set, the default order is to run the ESQL/C preprocessor on the program source file and pass the resulting file to the C language preprocessor and compiler. You can, however, compile an ESQL/C program source file in the following order:

1. Run the C preprocessor
2. Run the ESQL/C preprocessor
3. Run the C compiler and linker

To use a nondefault compilation order for a specific program, you can either give the program source file a **.ecp** extension, run the **-cp** option with the **esql** command on a program source file with a **.ec** extension, or set **CPFIRST**.

Set the **CPFIRST** environment variable to TRUE (uppercase only) to run the C preprocessor on all ESQL/C source files. The C preprocessor will run before the ESQL/C preprocessor on all ESQL/C source files in your environment, irrespective of whether the **-cp** option is passed to the **esql** command or the source files have the **.ec** or the **.ecp** extension.

To restore the default order on a system where the **CPFIRST** environment variable has been set to TRUE, you can set **CPFIRST** to FALSE. On UNIX systems that support the C shell, the following command has the same effect:

```
unsetenv CPFIRST
```

DBACCNOIGN

The **DBACCNOIGN** environment variable affects the behavior of the DB-Access utility if an error occurs under one of the following circumstances:

IDS

- You run DB-Access in nonmenu mode.
- You execute the LOAD command with DB-Access in menu mode. ♦

Set the **DBACCNOIGN** environment variable to 1 to roll back an incomplete transaction if an error occurs while you run the DB-Access utility under either of the preceding conditions.

```
setenv ──────────── DBACCNOIGN ──── 1 ────────┤
```

For example, assume DB-Access runs the following SQL commands:

```
DATABASE mystore
BEGIN WORK

INSERT INTO receipts VALUES (cust1, 10)
INSERT INTO receipt VALUES (cust1, 20)
INSERT INTO receipts VALUES (cust1, 30)

UPDATE customer
   SET balance =
      (SELECT (balance-60)
      FROM customer WHERE custid = 'cust1')
   WHERE custid = 'cust1

COMMIT WORK
```

In this example, one statement has a misspelled table name. The **receipt** table does not exist.

If your environment does not have **DBACCNOIGN** set, DB-Access inserts two records into the **receipts** table and updates the **customer** table. The decrease in the **customer** balance exceeds the sum of the inserted receipts.

If **DBACCNOIGN** is set to 1, messages appear, indicating that DB-Access rolled back all the INSERT and UPDATE statements. The messages also identify the cause of the error so that you can resolve the problem.

IDS

Load Statement Example

You can set **DBACCNOIGN** to protect data integrity during a LOAD statement, even if DB-Access runs the LOAD statement in menu mode.

Assume you execute the LOAD statement from the DB-Access SQL menu. Forty-nine rows of data load correctly, but the fiftieth row contains an invalid value that causes an error. If you set **DBACCNOIGN** to `1`, the database server does not insert the forty-nine previous rows into the database. If **DBACCNOIGN** is not set, the database server inserts the first forty-nine rows.

DBANSIWARN

Setting the **DBANSIWARN** environment variable indicates that you want to check for Informix extensions to ANSI-standard SQL syntax. Unlike most environment variables, you do not need to set **DBANSIWARN** to a value. You can set it to any value or to no value.

setenv —— DBANSIWARN ——|

If you set the **DBANSIWARN** environment variable for DB-Access, it is functionally equivalent to including the **-ansi** flag when you invoke DB-Access (or any Informix product that recognizes the **-ansi** flag) from the command line. If you set **DBANSIWARN** before you run DB-Access, any syntax-extension warnings are displayed on the screen within the SQL menu.

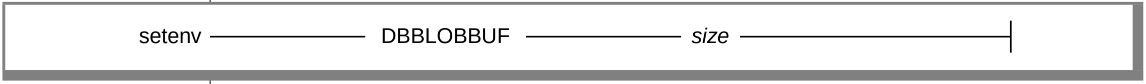
At runtime, the **DBANSIWARN** environment variable causes the sixth character of the **sqlwarn** array in the SQL Communication Area (SQLCA) to be set to `W` when a statement is executed that is recognized as including any Informix extension to the ANSI/ISO standard for SQL syntax. (For more information on SQLCA, see the *Informix ESQL/C Programmer's Manual*.)

After you set **DBANSIWARN**, Informix extension checking is automatic until you log out or unset **DBANSIWARN**. To turn off Informix extension checking, you can disable **DBANSIWARN** by entering the following command:

```
unsetenv DBANSIWARN
```

DBBLOBBUF

The **DBBLOBBUF** environment variable controls whether TEXT or BYTE data is stored temporarily in memory or in a file while being unloaded with the UNLOAD statement.



size	represents the maximum size of TEXT or BYTE data in kilobytes.

If a TEXT or BYTE (simple large object) data size is smaller than the default of 10 kilobytes or the setting of the **DBBLOBBUF** environment variable, it is temporarily stored in memory. If the TEXT or BYTE data is larger than the default or the setting of **DBBLOBBUF**, it is written to a temporary file. This environment variable applies to the UNLOAD command only.

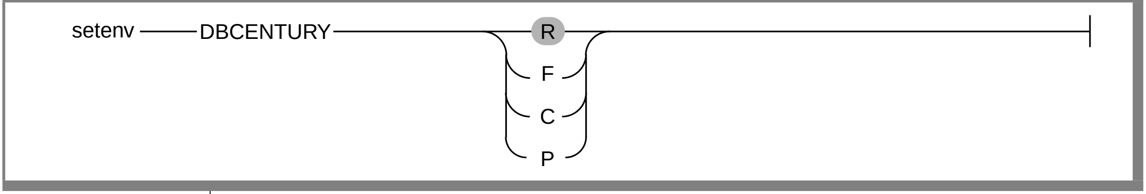
For instance, to set a buffer size of 15 kilobytes, set the **DBBLOBBUF** environment variable as the following example shows:

```
setenv DBBLOBBUF 15
```

In the example, any TEXT or BYTE data that is smaller than 15 kilobytes is stored temporarily in memory. TEXT or BYTE data larger than 15 kilobytes is stored temporarily in a file.

DBCENTURY

To avoid problems in expanding abbreviated years, applications should require entry of 4-digit years, and should always display years as four digits. The **DBCENTURY** environment variable specifies how to expand literal DATE and DATETIME values that are entered with abbreviated year values.



The R, P, F, and C settings choose algorithms for expanding two-digit years.

Setting	Algorithm
R = Current	Use the first two digits of the current year to expand the year value.
P = Past	Expanded dates are created by prefixing the abbreviated year value with “19” and “20.” Both dates are compared to the current date, and the most recent date that is earlier than the current date is used.
F = Future	Expanded dates are created by prefixing the abbreviated year value with “20” and “21.” Both dates are compared to the current date, and the earliest date that is later than the current date is used.
C = Closest	Expanded dates are created by prefixing the abbreviated year value with “19,” “20,” and “21.” These three dates are compared to the current date, and the date that is closest to the current date is used.

Settings are case sensitive, and no error is issued for invalid settings. If you enter “`f`” (for example), then the default (`R`) setting takes effect. The `P` and `F` settings cannot return the current date, which is not in the past or future.

When **DBCENTURY** is not set (or is set to `R`), the first two digits of the current year are used to expand the year value. For example, if today’s date is 09/30/2000, then the abbreviated date `12/31/99` expands to 12/31/2099, and the abbreviated date `12/31/00` expands to 12/31/2000.

Years entered as a single digit are prefixed with 0 and then expanded. Three-digit years are not expanded. Pad years earlier than 100 with leading zeros.

Examples of Expanding Year Values

The following examples illustrate how various settings of **DBCENTURY** cause abbreviated years to be expanded in DATE and DATETIME values.

DBCENTURY = P

```
Example data type: DATE
Current date: 4/6/2002
User enters: 1/1/1
Prefix with “19” expansion : 1/1/1901
Prefix with “20” expansion: 1/1/2001
Analysis: Both are prior to current date, but 1/1/2001 is closer
to current date.
```

DBCENTURY = F

```
Example data type: DATETIME year to month
Current date: 5/7/2005
User enters: 1-1
Prefix with “20” expansion: 2001-1
Prefix with “21” expansion: 2101-1
Analysis: Only date 2101-1 is after the current date, so it is
chosen.
```

DBCENTURY = C

```
Example data type: DATE
Current date: 4/6/2000
User enters: 1/1/1
Prefix with “19” expansion : 1/1/1901
Prefix with “20” expansion: 1/1/2001
Prefix with “21” expansion: 1/1/2101
Analysis: Here 1/1/2001 is closest to the current date, so it is
chosen.
```

DBCENTURY = R or DBCENTURY Not Set

```
Example data type: DATETIME year to month
Current date: 4/6/2000
User enters: 1-1
Prefix with “20” expansion: 2001-1

Example data type: DATE
Current date: 4/6/2003
User enters: 0/1/1
Prefix with “20” expansion: 2000/1
Analysis: In both examples, the Prefix with “20” algorithm is
used.
```

Important: *The effect of **DBCENTURY** depends on the current date from the system clock-calendar. Thus, 1/1/1, the abbreviated date the first example, would instead be expanded to 1/1/1901 if the current date were 1/1/2001 and **DBCENTURY** = P.*

Setting **DBCENTURY** does not affect Informix products when the locale specifies a non-Gregorian calendar, such as Hebrew or Islamic calendars. The leading digits of the current year are used for alternate calendar systems when the year is abbreviated.

Abbreviated Years and Expressions in Database Objects

When an expression in a database object (including a check constraint, fragmentation expression, SPL routine, trigger, or UDR) contains a literal DATE or DATETIME value in which the year has 1 or 2 digits, the database server evaluates the expression using the setting that **DBCENTURY** (and other relevant environment variables) had when the database object was created (or was last modified). If **DBCENTURY** has been reset to a new value, the new value is ignored when the abbreviated year is expanded.

For example, suppose a user creates a table and defines the following check constraint on a column named **birthdate**:

```
birthdate < '09/25/50'
```

The expression is interpreted according to the value of **DBCENTURY** when the constraint was defined. If the table that contains the **birthdate** column is created on 09/23/2000 and **DBCENTURY** `=C`, the check constraint expression is consistently interpreted as `birthdate < '09/25/1950'` when inserts or updates are performed on the **birthdate** column, regardless of any changes to the value of **DBCENTURY**. Even if different values of **DBCENTURY** are set when users perform inserts or updates on the **birthdate** column, the check constraint expression is interpreted according to the setting at the time when the check constraint was defined (or was last modified).

Database objects created on some earlier versions of Dynamic Server do not support the priority of creation-time settings.

For legacy objects to acquire this feature

1. Drop the objects.
2. Re-create them (or for fragmentation expressions, detach them and then reattach them).

After the objects are redefined, date literals within expressions of the objects will be interpreted according to the environment at the time when the object was created or was last modified. Otherwise, their behavior will depend on the runtime environment and might become inconsistent if this changes.

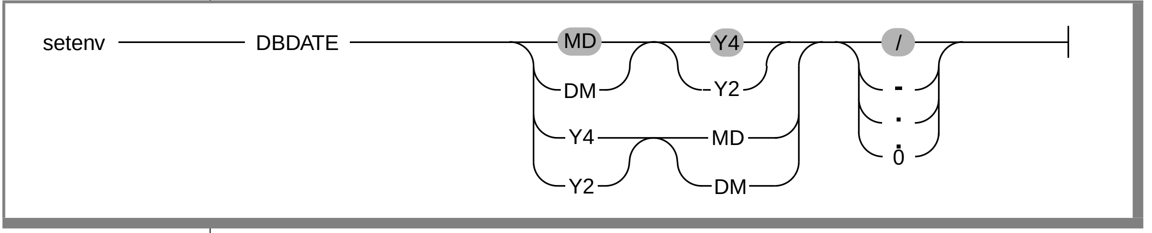
Administration of a database that includes a mix of legacy objects and new objects might become difficult because of differences between the new and the old behavior for evaluating date expressions. To avoid this, Informix recommends that you redefine any legacy objects.

The value of **DBCENTURY** and the current date are not the only factors that determine how the database server interprets DATE and DATETIME values. The **DBDATE**, **DBTIME**, **GL_DATE**, and **GL_DATETIME** environment variables can also influence how dates are interpreted. For information about **GL_DATE** and **GL_DATETIME**, see the *Informix Guide to GLS Functionality*.

Important: *The behavior of **DBCENTURY** for Dynamic Server and Extended Parallel Server is not backwards compatible.*

DBDATE

The **DBDATE** environment variable specifies the end-user formats of DATE values. On UNIX systems that use the C shell, set **DBDATE** with this syntax:



- . /	are characters that can appear as separators in a DATE format.
0	indicates that no separator is displayed between time units.
D, M	are characters that represent the day and the month.
Y2, Y4	are characters that represent the year and the precision of the year.

GLS

Some East Asian locales support additional syntax for era-based dates. For a description of era-based formats, see *Informix Guide to GLS Functionality*. ♦

DBDATE can specify the following attributes of the display format:

- The order of time units (the month, day, and year) in a date
- Whether the year appears with two digits (Y2) or four digits (Y4)
- The separator between the month, day, and year time units

For the U.S. English locale, the default for **DBDATE** is MDY4/, where M represents the month, D represents the day, Y4 represents a four-digit year, and slash (**/**) is the time-units separator (for example, 01/08/2002). Other valid characters for the separator are a hyphen (**-**), a period (**.**), or a zero (0). To indicate no separator, use the zero. The slash (**/**) is used by default if you attempt to specify a character other than a hyphen, period, or zero as a separator, or if you do not include any separator in the **DBDATE** specification.

The following table shows some examples for setting DBDATE:

DBDATE Setting	January 8, 2001, appears as:
MDY4/	01/08/2001
DMY2-	08-01-01
MDY4	01/08/2001
Y2DM.	01.08.01
MDY20	010801
Y4MD*	2001/01/08

Formats Y4MD* (because asterisk is not a valid separator) and MDY4 (with no separator defined) both display the default symbol (slash) as the separator.

Important: *If you use the Y2 format, the setting of the* ***DBCENTURY*** *environment variable can also affect how literal DATE values are evaluated in data entry.*

If **DBDATE** is not set on the client, any **DBDATE** setting on the database server overrides the MDY4/ default on the client. If **DBDATE** is set on the client, that value (rather than the setting on the database server) is used by the client.

Also, certain routines that Informix ESQL/C calls can use the **DBTIME** variable, rather than **DBDATE**, to set DATETIME formats to international specifications. For more information, see the discussion of the **DBTIME** environment variable in “DBTIME” on page 3-48 and the *Informix ESQL/C Programmer’s Manual*.

GLS

The setting of the **DBDATE** variable takes precedence over that of the **GL_DATE** environment variable, as well as over any default DATE format that **CLIENT_LOCALE** specifies. For information about **GL_DATE** and **CLIENT_LOCALE**, see the *Informix Guide to GLS Functionality*. ♦

End-user formats affect the following contexts:

- When you display DATE values, Informix products use the **DBDATE** environment variable to format the output.
- During data entry of DATE values, Informix products use the **DBDATE** environment variable to interpret the input.

For example, if you specify a literal DATE value in an INSERT statement, the database server expects this literal value to be compatible with the format that **DBDATE** specifies. Similarly, the database server interprets the date that you specify as input to the DATE( ) function in the format that **DBDATE** specifies.

DATE Expressions in Database Objects

When an expression in a database object (including a check constraint, fragmentation expression, SPL routine, trigger, or UDR) contains a literal DATE value, the database server evaluates the expression using the setting that **DBDATE** (or other relevant environment variables) had when the database object was created (or was last modified). If **DBDATE** has been reset to a new value, the new value is ignored when the literal DATE is evaluated.

For example, suppose **DBDATE** is set to `MDY2/` and a user creates a table with the following check constraint on the column **orderdate**:

```
orderdate < '06/25/98'
```

The date of the preceding expression is formatted according to the value of **DBDATE** when the constraint is defined. The check constraint expression is interpreted as `orderdate < '06/25/98'` regardless of the value of **DBDATE** during inserts or updates on the **orderdate** column. Suppose **DBDATE** is reset to `DMY2/` when a user inserts the value `'30/01/98'` into the **orderdate** column. The date value inserted uses the date format `DMY2/`, whereas the check constraint expression uses the date format `MDY2/`.

See the section "Abbreviated Years and Expressions in Database Objects" on page 3-31 for a discussion of legacy objects from earlier versions of Informix database servers that are always evaluated according to the runtime environment. That section describes how to redefine objects so that dates are interpreted according to the setting of **DBDATE** (and of other environment variables) at the time when the object was defined (or was last modified).

***Important:** The behavior of **DBDATE** for Dynamic Server and Extended Parallel Server is not backwards compatible.*

DBDELIMITER

The **DBDELIMITER** environment variable specifies the field delimiter used with the **dbexport** utility and with the LOAD and UNLOAD statements.

```
setenv ——— DBDELIMITER ——— 'delimiter' ———|
```

delimiter is the field delimiter for unloaded data files.

The *delimiter* can be any single character, except those in the following list:

- Hexadecimal digits (0 through 9, a through f, A through F)
- Newline or CTRL-J
- The backslash (\ ) symbol

The vertical bar (| = ASCII 124) is the default. To change the field delimiter to a plus (+) symbol, for example, you can set DBDELIMITER as follows:

```
setenv DBDELIMITER '+'
```

DBEDIT

The **DBEDIT** environment variable specifies the text editor to use with SQL statements and command files in DB-Access. If **DBEDIT** is set, the specified text editor is invoked automatically. If **DBEDIT** is not set, you are prompted to specify a text editor as the default for the rest of the session.

```
setenv ——— DBEDIT ——— editor ———|
```

editor is the name of the text editor you want to use.

For most UNIX systems, the default text editor is **vi**. If you use another text editor, be sure that it creates flat ASCII files. Some word processors in *document mode* introduce printer control characters that can interfere with the operation of your Informix product.

To specify the EMACS text editor, set **DBEDIT** by entering the following command:

```
setenv DBEDIT emacs
```

DBFLTMASK

The DB-Access utility displays the floating-point values of data types FLOAT, SMALLFLOAT, and DECIMAL(*p*) within a 14-character buffer. By default, DB-Access displays as many digits to the right of the decimal point as will fit into this character buffer. Therefore, the actual number of decimal digits that DB-Access displays depends on the size of the floating-point value.

To reduce the number of digits displayed to the right of the decimal point in floating-point values, set **DBFLTMASK** to the desired number of digits.

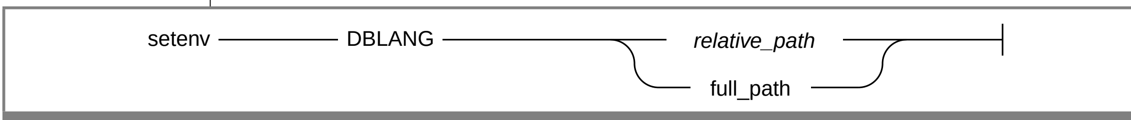
setenv —— DBFLTMASK —— *scale* ——|

scale is the number of decimal digits that you want the Informix client application to display in the floating-point values. Here *scale* must be smaller than 16, the default number of digits displayed.

If the floating-point value contains more digits to the right of the decimal than **DBFLTMASK** specifies, DB-Access rounds the value to the specified number of digits. If the floating-point value contains fewer digits to the right of the decimal, DB-Access pads the value with zeros. If you set **DBFLTMASK** to a value greater than can fit into the 14-character buffer, however, DB-Access rounds the value to the number of digits that can fit.

DBLANG

The **DBLANG** environment variable specifies the subdirectory of $INFORMIXDIR or the full pathname of the directory that contains the compiled message files that an Informix product uses.



relative_path is a subdirectory of $INFORMIXDIR.

full_path is the pathname to the compiled message files.

By default, Informix products put compiled messages in a locale-specific subdirectory of the $INFORMIXDIR**/msg** directory. These compiled message files have the file extension **.iem**. If you want to use a message directory other than $INFORMIXDIR**/msg**, where, for example, you can store message files that you create, you must perform the following steps:

To use a message directory other than $INFORMIXDIR/msg

1. Use the **mkdir** command to create the appropriate directory for the message files.

   You can make this directory under the directory **$INFORMIXDIR** or **$INFORMIXDIR/msg**, or you can make it under any other directory.
2. Set the owner and group of the new directory to **informix** and the access permission for this directory to 755.
3. Set the **DBLANG** environment variable to the new directory.

   If this directory is a subdirectory of **$INFORMIXDIR** or **$INFORMIXDIR/msg**, then you need only list the relative path to the new directory. Otherwise, you must specify the full pathname of the directory.
4. Copy the **.iem** files or the message files that you created to the new message directory that **$DBLANG** specifies.

   All the files in the message directory should have the owner and group **informix** and access permission 644.

Informix products that use the default U.S. English locale search for message files in the following order:

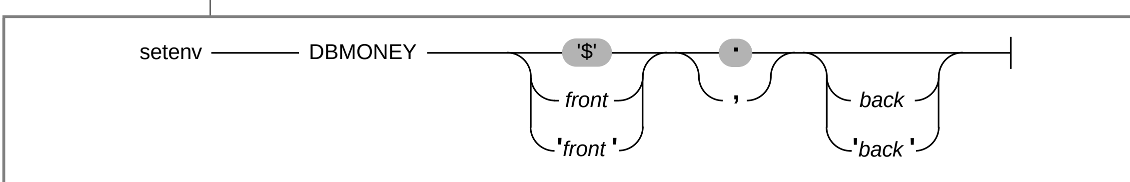
1. In **$DBLANG**, if **DBLANG** is set to a full pathname
2. In **$INFORMIXDIR/msg/$DBLANG**, if **DBLANG** is set to a relative pathname
3. In **$INFORMIXDIR/$DBLANG**, if **DBLANG** is set to a relative pathname
4. In **$INFORMIXDIR/msg/en_us/0333**
5. In **$INFORMIXDIR/msg/en_us.8859-1**
6. In $INFORMIXDIR**/msg**
7. In **$INFORMIXDIR/msg/english**

GLS

For more information on search paths for messages, see the description of **DBLANG** in the *Informix Guide to GLS Functionality*. ♦

DBMONEY

The **DBMONEY** environment variable specifies the display format of values in columns of SMALLFLOAT, FLOAT, DECIMAL, or MONEY data types, and of complex data types derived from any of these data types.



$ is a currency symbol that precedes MONEY values in the default locale if no other *front* symbol is specified, or if **DBMONEY** is not set.

, *or* . is a comma or period (the default) that separates the integral part from the fractional part of the FLOAT, DECIMAL, or MONEY value.

back is a currency symbol that follows the MONEY value. The *back* symbol can be up to seven characters and can contain any character that the locale supports, except a digit, a comma (,), or a period (.) symbol.

front is a currency symbol that precedes the MONEY value. The *front* symbol can be up to seven characters and can contain any character that the locale supports, except a digit, a comma, or a period.

If you specify any character that is not a letter of the alphabet for *front* or *back*, you must enclose the *front* or *back* setting between single quotation (') marks.

When you display MONEY values, Informix products use the **DBMONEY** setting to format the output. **DBMONEY** has no effect, however, on the internal format of data values that are stored in columns of the database.

If you do not set **DBMONEY**, then MONEY values for the default locale, U.S. English, are formatted with a dollar sign ($) that precedes the MONEY value, a period (.) that separates the integral from the fractional part of the MONEY value, and no *back* symbol. For example, `100.50` is formatted as `$100.50`.

GLS

Suppose you want to represent MONEY values as DM *(*deutsche mark*)* units, using the currency symbol `DM` and comma (,) as the decimal separator. Enter the following command to set the **DBMONEY** environment variable:

```
setenv DBMONEY DM,
```

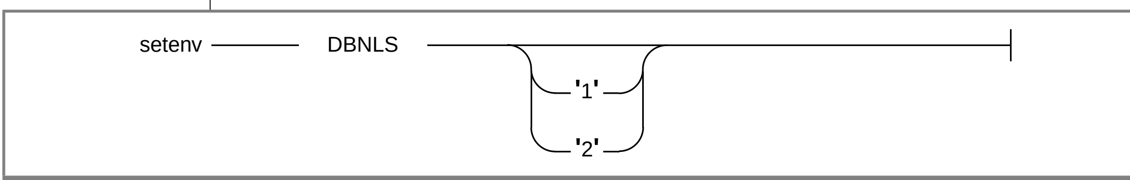
Here `DM` is the *front* currency symbol that precedes the MONEY value, and a comma separates the integral from the fractional part of the MONEY value. As a result, the value `100.50` is displayed as `DM100,50`.

For more information about how **DBMONEY** formats MONEY values in nondefault locales, see the *Informix Guide to GLS Functionality*. ♦

DBNLS

The **DBNLS** environment variable specifies whether automatic data type conversion is supported between NCHAR and NVARCHAR database columns and CHAR and VARCHAR variables (respectively) of the client systems.

Global Language Support (GLS) does not require the **DBNLS** environment variable. But Dynamic Server databases continue to support the legacy behavior of **DBNLS**, which is useful in applications for databases that include tables with NCHAR or NVARCHAR columns.



For UNIX systems that use the C shell, the following command line enables client applications such as DB-Access, Informix-SQL, Informix-4GL, Dynamic 4GL, and embedded-SQL applications such as ESQL/C or ESQL/COBOL to convert automatically between CHAR and VARCHAR variables of the client application and NCHAR and NVARCHAR columns of the database:

```
setenv DBNLS 1
```

This setting also supports the automatic conversion of values retrieved from NCHAR columns into CHAR variables, and the conversion of NVARCHAR column values into VARCHAR variables. Similarly, when **DBNLS** = 1, character strings stored as CHAR variables can be inserted into NCHAR columns, and character strings stored as VARCHAR variables can be inserted into NVARCHAR database columns.

To support these features, **DBNLS** must also be set to 1 on the client system. This setting also enables the client system to display dates, numbers, and currency values in formats specified on the client locale.

Conversely, each of the following command lines disables automatic conversion between CHAR and VARCHAR variables of the client application and NCHAR and NVARCHAR columns of the database, and also prevents Dynamic Server from using the locale files of the client system:

```
setenv DBNLS
```

```
unsetenv DBNLS
```

On UNIX systems that use the C shell, either of these commands disables automatic conversion to and from NCHAR and NVARCHAR data values (by setting no value for **DBNLS**).

Another possible setting for **DBNLS** is 2. If you enter at the command line

```
setenv DBNLS 2
```

then automatic data type conversion between NCHAR and CHAR and between NVARCHAR and VARCHAR is supported (if the client system has **DBNLS** set to 1 or 2), but the database server can have a different locale from the client system.

IDS

DBONPLOAD

The **DBONPLOAD** environment variable specifies the name of the database that the **onpload** utility of the High-Performance Loader (HPL) uses. If **DBONPLOAD** is set, **onpload** uses the specified name as the name of the database; otherwise, the default name of the database is **onpload**.

setenv —— DBONPLOAD —— *dbname* ——|

dbname specifies the name of the database that the **onpload** utility uses.

For example, to specify the name **load_db** as the name of the database, enter the following command:

```
setenv DBONPLOAD load_db
```

For more information, see the *Guide to the High-Performance Loader*.

DBPATH

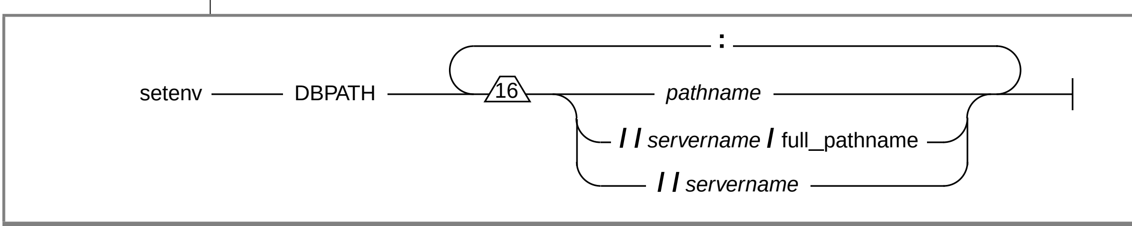
The **DBPATH** environment variable identifies database servers that contain databases. DBPATH can also specify a list of directories (in addition to the current directory) in which DB-Access looks for command scripts (**.sql** files).

The CONNECT, DATABASE, START DATABASE, and DROP DATABASE statements use **DBPATH** to locate the database under two conditions:

- If the location of a database is not explicitly stated
- If the database cannot be located in the default server

The CREATE DATABASE statement does not use **DBPATH**.

To add a new **DBPATH** entry to existing entries, see "Modifying an Environment-Variable Setting" on page 3-10.



full_pathname is the full path, from **root**, of a directory where **.sql** files are stored.

pathname is the valid relative path of a directory where **.sql** files are stored.

servername is the name of an Informix database server where databases are stored. You cannot reference database files with a *servername*.

DBPATH can contain up to 16 entries. Each entry must be less than 128 characters. In addition, the maximum length of **DBPATH** depends on the hardware platform on which you set **DBPATH**.

When you access a database with the CONNECT, DATABASE, START DATABASE, or DROP DATABASE statement, the search for the database is done first in the directory and / or database server specified in the statement. If no database server is specified, the default database server that was specified by the **INFORMIXSERVER** environment variable is used.

If the database is not located during the initial search, and if **DBPATH** is set, the database servers and / or directories in **DBPATH** are searched for in the specified database. These entries are searched in the same order in which they are listed in the **DBPATH** setting.

Using DBPATH with DB-Access

If you use DB-Access and select the **Choose** option from the SQL menu without having already selected a database, you see a list of all the **.sql** files in the directories listed in your **DBPATH**. Once you select a database, the **DBPATH** is not used to find the **.sql** files. Only the **.sql** files in the current working directory are displayed.

Searching Local Directories

Use a pathname without a database server name to search for **.sql** scripts on your local computer. In the following example, the **DBPATH** setting causes DB-Access to search for the database files in your current directory and then in Joachim's and Sonja's directories on the local computer:

```
setenv DBPATH /usr/joachim:/usr/sonja
```

As the previous example shows, if the pathname specifies a directory name but not a database server name, the directory is sought on the computer that runs the default database server that the **INFORMIXSERVER** specifies; see "INFORMIXSERVER" on page 3-68. For instance, with the previous example, if **INFORMIXSERVER** is set to **quality**, the **DBPATH** value is *interpreted*, as the following example shows, where the double slash precedes the database server name:

```
setenv DBPATH //quality/usr/joachim://quality/usr/sonja
```

Searching Networked Computers for Databases

If you use more than one database server, you can set **DBPATH** explicitly to contain the database server and/or directory names that you want to search for databases. For example, if **INFORMIXSERVER** is set to **quality** but you also want to search the **marketing** database server for **/usr/joachim**, set **DBPATH** as the following example shows:

```
setenv DBPATH //marketing/usr/joachim:/usr/sonja
```

Specifying a Servername

You can set **DBPATH** to contain only database server names. This feature allows you to locate only databases; you cannot use it to locate command files.

The database administrator must include each database server mentioned by **DBPATH** in the **$INFORMIXDIR/etc/sqlhosts** file. For information on communication-configuration files and dbservernames, see your *Administrator's Guide* and the *Administrator's Reference.*

For example, if **INFORMIXSERVER** is set to **quality**, you can search for a database first on the **quality** database server and then on the **marketing** database server by setting **DBPATH,** as the following example shows:

```
setenv DBPATH //marketing
```

If you use DB-Access in this example, the names of all the databases on the **quality** and **marketing** database servers are displayed with the **Select** option of the DATABASE menu.

DBPRINT

The **DBPRINT** environment variable specifies the default printing program.

setenv —— DBPRINT —— *program* ——|

program is any command, shell script, or UNIX utility that produces standard ASCII output.

If you do not set **DBPRINT**, the default *program* is found in one of two places:

- For most BSD UNIX systems, the default program is **lpr**.
- For UNIX System V, the default program is usually **lp**.

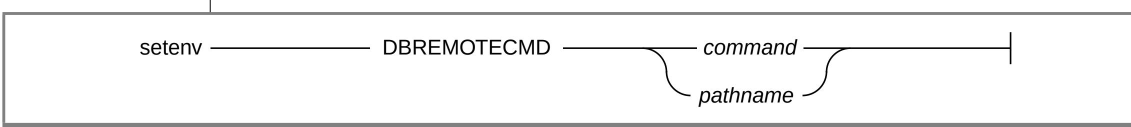
Enter the following command to set the **DBPRINT** environment variable to specify **myprint** as the print program:

```
setenv DBPRINT myprint
```

UNIX

DBREMOTECMD

Set the **DBREMOTECMD** environment variable to override the default remote shell to perform remote tape operations with the database server. You can set **DBREMOTECMD** to a simple command or to a full pathname.



command is a command to override the default remote shell.

pathname is a pathname to override the default remote shell.

If you do not specify the full pathname, the database server searches your **PATH** for the specified *command*. Informix highly recommends use of the full pathname syntax on interactive UNIX platforms to avoid problems with similarly named programs in other directories and possible confusion with the *restricted shell* (**/usr/bin/rsh**).

The following command sets **DBREMOTECMD** for a simple command name:

```
setenv DBREMOTECMD rcmd
```

The next command to set **DBREMOTECMD** specifies a full pathname:

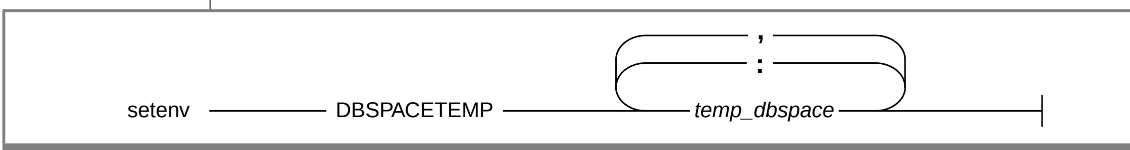
```
setenv DBREMOTECMD /usr/bin/remsh
```

For more information on **DBREMOTECMD**, see the discussion in your *Archive and Backup Guide* about how to use remote tape devices with your database server for archives, restores, and logical-log backups.

DBSPACETEMP

The **DBSPACETEMP** environment variable specifies the dbspaces in which temporary tables are built

You can list dbspaces, separated by colon (:) or comma (,) symbols, to spread temporary space across any number of disks.



temp_dbspace is the name of a valid existing temporary dbspace.

DBSPACETEMP overrides any default dbspaces that the DBSPACETEMP parameter specifies in the configuration file of the database server.

Important: *The dbspaces that you list in **DBSPACETEMP** must be composed of chunks that are allocated as raw UNIX devices.*

For example, the following command to set the **DBSPACETEMP** environment variable specifies three dbspaces for temporary tables:

```
setenv DBSPACETEMP sorttmp1:sorttmp2:sorttmp3
```

Separate the dbspace entries with either colons or commas. The number of dbspaces is limited by the maximum size of the environment variable, as defined by your operating system. Your database server does not create a dbspace specified by the environment variable if the dbspace does not exist.

The two classes of temporary tables are *explicit* temporary tables that the user creates and *implicit* temporary tables that the database server creates. Use **DBSPACETEMP** to specify the dbspaces for both types of temporary tables.

If you create an explicit temporary table with the CREATE TEMP TABLE statement and do not specify a dbspace for the table either in the IN *dbspace* clause or in the FRAGMENT BY clause, the database server uses the settings in **DBSPACETEMP** to determine where to create the table.

If you create an explicit temporary table with the SELECT INTO TEMP statement, the database server uses the settings in **DBSPACETEMP** to determine where to create the table. If **DBSPACETEMP** is not set, the database server uses the ONCONFIG parameter DBSPACETEMP. If this parameter is not set, the database server creates the explicit temporary table in the same dbspace where the database resides.

The database server creates implicit temporary tables for its own use while executing join operations, SELECT statements with the GROUP BY clause, SELECT statements with the ORDER BY clause, and index builds. When it creates these implicit temporary tables, the database server uses disk space for writing the temporary data, in the following order:

UNIX

1. The operating-system directory or directories that the environment variable **PSORT_DBTEMP** specifies, if it is set. ♦
2. The dbspace or dbspaces that the environment variable **DBSPACETEMP** specifies, if it is set.
3. The dbspace or dbspaces that the ONCONFIG parameter DBSPACETEMP specifies.
4. The operating-system file space in **/tmp** (UNIX) or **%temp%** (Windows).

Important: *If the **DBSPACETEMP** environment variable is set to an invalid value, the database server defaults to the root dbspace for explicit temporary tables and to* **/tmp** *for implicit temporary tables, not to the DBSPACETEMP configuration parameter. In this situation, the database server might fill **/tmp** to the limit and eventually bring down the database server or kill the file system.*

DBTEMP

The **DBTEMP** environment variable is used by DB-Access and Informix Enterprise Gateway products. **DBTEMP** resembles **DBSPACETEMP** in its functionality, specifying the directory in which to place temporary files and temporary tables.

```
setenv ——— DBTEMP ——— pathname ———|
```

pathname is the full pathname of the directory for temporary files and temporary tables.

On UNIX systems, temporary files are created in **/tmp** if **DBTEMP** is not set.

The following example sets **DBTEMP** to the pathname **usr/magda/mytemp** for UNIX systems that use the C shell:

```
setenv DBTEMP usr/magda/mytemp
```

Important: ***DBTEMP** can point to an NFS-mounted directory only if the vendor of that NFS device is certified by Informix. For information about NFS products that you can use to mount a storage space for an Informix database server, see the product compatibility information on the Informix web site at http://www.informix.com/idn-secure/pacc/prodcompat.html*

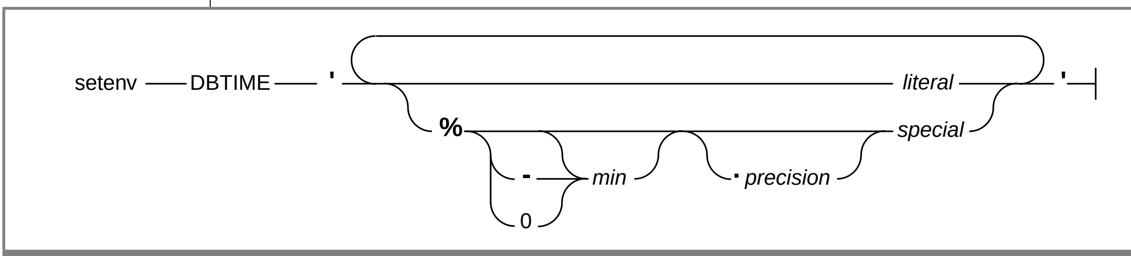
For DB-Access to work correctly on Windows platforms, **DBTEMP** should be set to **$INFORMIXDIR/infxtmp**.

If you do not set **DBTEMP**, temporary files are created in **/tmp**. If **DBTEMP** is not set, temporary tables are created in the directory of the database (that is, the **.dbs** directory).

DBTIME

The **DBTIME** environment variable specifies a display and data-entry format for DATETIME values. Like **DBDATE**, **GL_DATE**, or **GL_DATETIME**, this environment variable controls only the character-string representation of data values; it cannot change the internal storage format of DATETIME columns.

You can specify a non-default format for DATETIME values in input and output operations by using **DBTIME** to define a formatting mask. This quoted string can include literal characters as well as placeholders for the values of individual time units and other elements of a DATETIME value.



literal is a literal white space or any printable character.

min is a literal integer, setting the minimum number of characters in the substring for the value that *special* specifies.

precision is the number of digits for the value of any time unit, or the maximum number of characters in the name of a month.

special is one of the placeholder characters that are listed on page 3-50.

These terms and symbols are described in the pages that follow.

DBTIME takes effect only when you call certain Informix ESQL/C DATETIME routines. (For details, see the *Informix ESQL/C Programmer's Manual*.) If **DBTIME** is not set, the behavior of these Informix ESQL/C DATETIME routines is undefined, and `"YYYY-MM-DD hh:mm:ss.fffff"` is the default display and input format for DATETIME YEAR TO FRACTION(5) literal values in the default locale.

Here the percentage (%) symbol gives special significance to the *special* placeholder symbol that follows. Without a preceding % symbol, any character within the formatting mask is interpreted as a literal character, even if it is the same character as one of the placeholder characters in the following list. Note also that the *special* placeholder symbols are case sensitive.

The following characters within a **DBTIME** format string are placeholders for time units (or for other features) within a DATETIME value.

%b	is replaced by the abbreviated month name.
%B	is replaced by the full month name.
%d	is replaced by the day of the month as a decimal number [01,31].
%F*n*	is replaced by the value of the fraction of a second with precision that the integer *n* specifies. The default value of *n* is 2; the range of *n* is 0 £ *n* £ 5.
%H	is replaced by the hour (24-hour clock).
%I	is replaced by the hour (12-hour clock).
%M	is replaced by the minute as a decimal number [00,59].
%m	is replaced by the month as a decimal number [01,12].
%p	is replaced by A.M. or P.M. (or the equivalent in the locale file).
%S	is replaced by the second as a decimal number [00,59].
%y	is replaced by the year as a four-digit decimal number. If the user enters a two-digit value, this value is expanded to 4 digits according to the setting of the **DBCENTURY** environment variable. If **DBCENTURY** is not set, then the string `19` is used by default for the first two digits.
%Y	is replaced by the year as a four-digit decimal number. User must enter a four-digit value.
%%	is replaced by % (to allow % in the format string).

For example, consider how to convert a DATETIME YEAR TO SECOND to the following ASCII string format:

```
Mar 21, 2001 at 16 h 30 m 28 s
```

Set **DBTIME** as the following command line (for the C shell) shows:

```
setenv DBTIME '%b %d, %Y at %H h %M m %S s'
```

The default **DBTIME** produces the conventional ANSI SQL string format that the following line shows:

```
2001-03-21 16:30:28
```

You can set the default **DBTIME** as the following example shows:

```
setenv DBTIME '%Y-%m-%d %H:%M:%S'
```

An optional field width and precision specification (*w.p*) can immediately follow the percent (%) character. It is interpreted as follows:

w Specifies the minimum field width. The value is right-justified with spaces on the left.

-*w* Specifies the minimum field width. The value is left-justified with spaces on the right.

0*w* Specifies the minimum field width. The value is right-justified and padded with zeros on the left.

p Specifies the number of digits to appear for `d`, `H`, `I`, `m`, `M`, `S`, `y`, and `Y` time unit values, or the maximum number of characters to use in `b` and `B` month names.

The following limitations apply to field-width and precision specifications:

- If the data value supplies fewer digits than *precision* specifies, the value is padded with leading zeros.
- If a data value supplies more characters than *precision* specifies, excess characters are truncated on the right.
- If no field width nor precision is specified for `d`, `H`, `I`, `m`, `M`, `S`, or `y` placeholders, a default of `0.2` is used. A default of `0.4` is used for `Y` placeholders.
- A *precision* specification is significant only when converting a DATETIME value to an ASCII string, but not vice versa.

The `F` placeholder does not support this field-width and precision syntax.

For formatting DATE values, see the discussion of **DBDATE** on page 3-32.

GLS

In East Asian locales that support era-based dates, **DBTIME** can also specify Japanese or Taiwanese eras. See *Informix Guide to GLS Functionality* for details of additional placeholder symbols for setting **DBTIME** to display era-based DATETIME values, and for descriptions of the **GL_DATETIME** and **GL_DATE** environment variables. ♦

DBUPSPACE

The **DBUPSPACE** environment variable lets you specify and constrain the amount of system disk space that the UPDATE STATISTICS statement can use when trying to simultaneously construct multiple column distributions.

```
setenv ────── DBUPSPACE ────── max ──────────────┤
```

max is a positive integer, specifying the maximum disk space (in kilobytes).

For example, to set **DBUPSPACE** to 2,500 kilobytes, enter this command:

```
setenv DBUPSPACE 2500
```

After you set this value, the database server can use no more than 2,500 kilobytes of disk space during the execution of an UPDATE STATISTICS statement. If a table requires 5 megabytes of disk space for sorting, then UPDATE STATISTICS accomplishes the task in two passes; the distributions for one half of the columns are constructed with each pass.

If you do not set **DBUPSPACE**, the default is a megabyte (= 1,024 kilobytes). If you attempt to set **DBUPSPACE** to any value less than 1,024 kilobytes, it is automatically set to 1,024 kilobytes, but no error message is returned. If this value is not large enough to allow more than one distribution to be constructed at a time, at least one distribution is done, even if the amount of disk space required to do this is more than what **DBUPSPACE** specifies.

DEFAULT_ATTACH

The **DEFAULT_ATTACH** environment variable supports the legacy behavior of Version 7.x of Dynamic Server, which required that only nonfragmented B-tree indexes on nonfragmented tables can be attached.

```
setenv ──────────── DEFAULT_ATTACH ──── 1 ────────┤
```

If **DEFAULT_ATTACH** is set to 1, then all other indexes, including R-trees and UDT indexes, must be detached. (An attached index is one that is created without specifying a storage clause.)

If **DEFAULT_ATTACH** is not set, then the CREATE INDEX statement creates detached indexes by default. This release of Dynamic Server can support attached indexes that were created by Version 7.x of Dynamic Server.

Important: ***DEFAULT_ATTACH*** *might not continue to be supported in future releases of Informix database servers. It is not recommended that you develop new database applications that depend on this deprecated feature.*

DELIMIDENT

The **DELIMIDENT** environment variable specifies that strings enclosed between double quotation (") marks are delimited database identifiers.

```
setenv ──── DELIMIDENT ──────────────────────────┤
```

No value is required; **DELIMIDENT** takes effect if it exists, and it remains in effect while it is on the list of environment variables.

Delimited identifiers can include white space (such as "**Vitamin E**") or can be identical to reserved keywords, (such as "**TABLE**" or "**USAGE**"). You can also use them to declare database identifiers that contain characters outside the default character set for SQL identifiers (such as "**Column #6**"). In the default locale, this set consists of letters, digits, and the underscore (_) symbol.

You cannot, however, use **DELIMIDENT** to declare storage identifiers that contain characters outside the default SQL character set. *Database identifiers* (also called *SQL identifiers*) are names for database objects, such as tables and columns. *Storage identifiers* are names for storage objects, such as dbspaces, blobspaces, and sbspaces (smart blob spaces).

Delimited identifiers are case sensitive.To use delimited identifiers, applications in ESQL/C must set **DELIMIDENT** at compile time and at runtime.

Warning: *If **DELIMIDENT** is not already set, you should be aware that setting it can cause the failure of existing **.sql** scripts or client applications that use double (") quotation marks in contexts other than delimiting SQL identifiers, such as delimiters of string literals. You must use single (') rather than double quotation marks for delimited constructs that are not SQL identifiers if **DELIMIDENT** is set.*

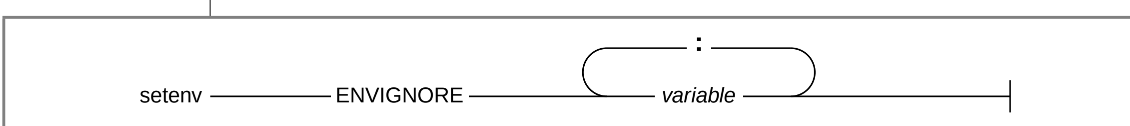
On UNIX systems that use the C shell and on which **DELIMIDENT** has been set, you can disable this feature (which causes anything between double quotes to be interpreted as an SQL identifier) by the command:

```
unsetenv  DELIMIDENT
```

UNIX

ENVIGNORE

The **ENVIGNORE** environment variable can deactivate specified environment variable settings in the common (shared) and private environment-configuration files, **informix.rc** and **.informix** respectively.



variable is the name of an environment variable to be deactivated.

Use colon (**:**) symbols between consecutive *variable* names. For example, to ignore the **DBPATH** and **DBMONEY** entries in the environment-configuration files, enter the following command:

```
setenv ENVIGNORE DBPATH:DBMONEY
```

The common environment-configuration file is stored in **$INFORMIXDIR/etc/informix.rc**. The private environment-configuration file is stored in the user's home directory as **.informix**. For information on creating or modifying an environment-configuration file, see "Setting Environment Variables in a Configuration File" on page 3-8.

ENVIGNORE itself cannot be set in an environment-configuration file.

FET_BUF_SIZE

The **FET_BUF_SIZE** environment variable can override the default setting for the size of the fetch buffer for all data types except BYTE and TEXT values.

```
setenv ——— FET_BUF_SIZE ——— size ———|
```

size is a positive integer that is larger than the default buffer size, but no greater than 32,767, specifying the size (in bytes) of the fetch buffer that holds data retrieved by a query.

For example, to set a buffer size to `5,000` bytes on a UNIX system that uses the C shell, set the **FET_BUF_SIZE** environment variable by entering the following command:

```
setenv FET_BUF_SIZE 5000
```

When **FET_BUF_SIZE** is set to a valid value, the new value overrides the default value (or any previously set value of **FET_BUF_SIZE**). The default setting for the fetch buffer is dependent on row size.

The processing of BYTE and TEXT values is not affected by **FET_BUF_SIZE**.

No error is raised if **FET_BUF_SIZE** is set to a value that is less than the default size or that is out of the range of SMALLINT values. In these cases, however, the invalid fetch buffer size is ignored, and the default size is in effect.

If you set **FET_BUF_SIZE** to a valid value, that value is in effect for the local database server, as well as for any remote database server from which you retrieve rows through a distributed query in which the local server is the coordinator and the remote database is subordinate. The greater the size of the buffer, the more rows can be returned, and the less frequently the client application must wait while the database server returns rows. A large buffer can improve performance by reducing the overhead of filling the client-side buffer.

XPS

IFMX_SMLTBL_BROADCAST_SIZE

The **IFMX_SMLTBL_BROADCAST_SIZE** environment variable setting on the database server determines the threshold size of tables that are used in Small Table Broadcast when the table size exceeds 128 kilobytes.

```
setenv ——— IFMX_SMLTBL_BROADCAST_SIZE ——— n ———|
```

n is a positive integer, representing the size of the table (in kilobytes).

Important: *Query performance can suffer if **IFMX_SMLTBL_BROADCAST_SIZE** is set beyond a certain table size. The recommended upper limit on table size depends on your computer and on the configuration of your database server.*

For more information about the **IFMX_SMLTBL_BROADCAST_SIZE** environment variable, see your XPS documentation.

IFX_DEF_TABLE_LOCKMODE

The **IFX_DEF_TABLE_LOCKMODE** environment variable can specify the default lock mode for database tables that are subsequently created without explicitly specifying the LOCKMODE PAGE or LOCKMODE ROW keywords. This feature is convenient if you need to create several tables of the same lock mode. UNIX systems that use the C shell support the following syntax:

```
setenv ——IFX_DEF_TABLE_LOCKMODE——+—PAGE—+———|
                                 +—ROW——+
```

PAGE	The default lock mode is page-level granularity.
ROW	The default lock mode is row-level granularity.

Similar functionality is available by setting the DEF_TABLE_LOCKMODE parameter of the ONCONFIG file to PAGE or ROW. When a table is created or modified, any conflicting lock mode specifications are resolved according to the following descending (highest to lowest) order of precedence:

1. Explicit LOCKMODE specification of CREATE TABLE or ALTER TABLE.
2. **IFX_DEF_TABLE_LOCKMODE** environment variable setting.
3. DEF_TABLE_LOCKMODE parameter setting in the ONCONFIG file.
4. The system default lock mode (= PAGE mode).

To make the DEF_TABLE_LOCKMODE setting the default mode (or to restore the system default, if DEF_TABLE_LOCKMODE is not set) use the command:

```
unsetenv IFX_DEF_TABLE_LOCKMODE
```

If **IFX_DEF_TABLE_LOCKMODE** is set in the environment of the database server before starting **oninit**, then its scope is all sessions of the database server (just as if DEF_TABLE_LOCKMODE were set in the ONCONFIG file).

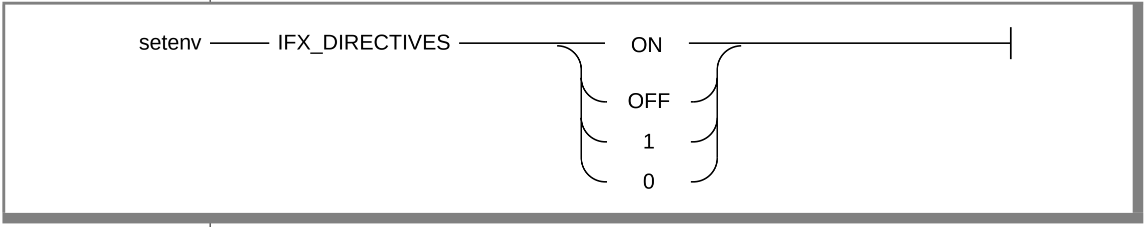
If **IFX_DEF_TABLE_LOCKMODE** is set in the shell, or in the **$HOME/.informix** or **$INFORMIXDIR/etc/informix.rc** files, then the scope is restricted to the current session (if you set it in the shell) or to the individual user.

Important: *This has no effect on existing tables. If you specify ROW as the lock mode, the database will use this to restore, recover, or copy data. For tables that were created in PAGE mode, this might cause lock-table overflow or performance degradation.*

IFX_DIRECTIVES

The **IFX_DIRECTIVES** environment variable setting determines whether the optimizer allows query optimization directives from within a query. The **IFX_DIRECTIVES** environment variable is set on the client.

You can use either `ON` and `OFF` or `1` and `0` to set the environment variable.



ON	Optimizer directives accepted
OFF	Optimizer directives not accepted
1	Optimizer directives accepted
0	Optimizer directives not accepted

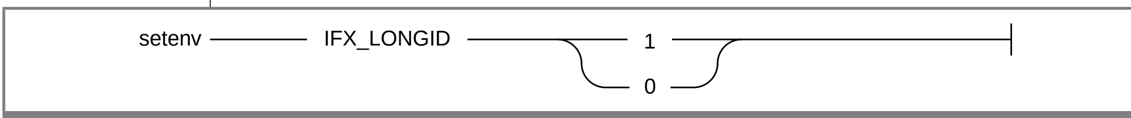
The setting of the **IFX_DIRECTIVES** environment variable overrides the value of the DIRECTIVES configuration parameter that is set for the database server. If the **IFX_DIRECTIVES** environment variable is not set, however, then all client sessions will inherit the database server configuration for directives that the ONCONFIG parameter DIRECTIVES determines. The default setting for the **IFX_DIRECTIVES** environment variable is `ON`.

For more information about the DIRECTIVES parameter, see the *Administrator's Reference*. For more information on the performance impact of directives, see your *Performance Guide*.

IDS

IFX_LONGID

The **IFX_LONGID** environment variable setting and the version number of the client application determine whether a given client application is capable of handling long identifiers. (Older versions of Informix databases restricted SQL identifiers to 18 or fewer bytes; *long identifiers* can have up to 128 bytes.) Valid **IFX_LONGID** values are `1` and `0`.



1 Client supports long identifiers

0 Client cannot support long identifiers

When **IFX_LONGID** is set to zero, applications display only the first 18 bytes of long identifiers, without indicating (by +) that truncation has occurred.

If **IFX_LONGID** is unset or is set to a value other than `1` or `0`, the determination is based on the internal version of the client application. If the version is not less than 9.0304, the client is considered capable of handling long identifiers. Otherwise, the client application is considered incapable.

The **IFX_LONGID** setting overrides the internal version of the client application. If the client cannot handle long identifiers despite a newer version number, set **IFX_LONGID** to `0`. If the client version can handle long identifiers despite an older version number, set **IFX_LONGID** to `1`.

If you set **IFX_LONGID** on the client, the setting affects only that client. If you bring up the database server with **IFX_LONGID** set, all client applications use that setting by default. If **IFX_LONGID** is set to different values on the client and on the database server, however, the client setting takes precedence.

Important: *ESQL executables that have been built with the* ***-static*** *option using the* ***libos.a*** *library version that does not support long identifiers cannot use the* ***IFX_LONGID*** *environment variable. You must recompile such applications with the new* ***libos.a*** *library that includes support for long identifiers. Executables that use shared libraries (no* ***-static*** *option) can use* ***IFX_LONGID*** *without recompilation provided that they use the new* ***libifos.so*** *that provides support for long identifiers. For details, see your ESQL product manual.*

UNIX

IFX_NETBUF_PVTPOOL_SIZE

The **IFX_NETBUF_PVTPOOL_SIZE** environment variable specifies the maximum size of the free (unused) private network buffer pool for each database server session.

```
setenv ——— IFX_NETBUF_PVTPOOL_SIZE ——— count ———|
```

count is an integer specifying the number of units (buffers) in the pool.

The default size is 1 buffer. If **IFX_NETBUF_PVTPOOL_SIZE** is set to 0, then each session obtains buffers from the free global network buffer pool. You must specify the value in decimal form.

IFX_NETBUF_SIZE

The **IFX_NETBUF_SIZE** environment variable lets you configure the network buffers to the optimum size. It specifies the size of all network buffers in the free (unused) global pool and the private network buffer pool for each database server session.

```
setenv ——— IFX_NETBUF_SIZE ——— size ———|
```

size is an integer specifying the size (in bytes) for one network buffer.

The default size is 4 kilobytes (4,096 bytes). The maximum size is 64 kilobytes (65,536 bytes) and the minimum size is 512 bytes. You can specify the value in hexadecimal or decimal form.

Tip: *You cannot set a different size for each session.*

IDS

IFX_UPDDESC

You must set the **IFX_UPDDESC** environment variable at execution time before you can do a DESCRIBE of an UPDATE statement.

```
setenv ──────── IFX_UPDDESC ──────── value ────────┤
```

value is any non-NULL value.

A NULL value (here meaning that **IFX_UPDDESC** is not set) disables the DESCRIBE-for-UPDATE feature. Any non-NULL value enables the feature.

IMCADMIN

The **IMCADMIN** environment variable supports the **imcadmin** administrative tool by specifying the name of a database server through which **imcadmin** can connect to MaxConnect. For **imcadmin** to operate correctly, you must set IMCADMIN before you use an Informix product.

```
setenv ────── IMCADMIN ────── dbservername ────────┤
```

dbservername is the name of a database server.

Here *dbservername* must listed in the **sqlhosts** file on the computer where the MaxConnect runs. MaxConnect uses this setting to obtain the following connectivity information from the **sqlhosts** file:

- Where the administrative listener port must be established
- The network protocol that the specified database server uses
- The host name of the system where the specified database server resides

You cannot use the **imcadmin** tool unless **IMCADMIN** is set to a valid database server name.

For more information about using **IMCADMIN**, refer to *Guide to Informix MaxConnect*.

IMCCONFIG

The **IMCCONFIG** environment variable specifies a nondefault filename, and optionally a pathname, for the MaxConnect configuration file. On UNIX systems that support the C shell, this variable can be set by the following command.

```
setenv ——— IMCCONFIG ——— pathname ———┤
```

pathname is a full pathname or a simple filename.

When the setting is a filename that is not qualified by a full pathname, MaxConnect searches for the specified file in the **$INFORMIXDIR/etc/** directory. Thus, if you set **IMCCONFIG** to **IMCconfig.imc2**, MaxConnect searches for **$INFORMIXDIR/etc/IMCconfig.imc2** as its configuration file.

If the **IMCCONFIG** environment variable is not set, MaxConnect searches by default for **$INFORMIXDIR/etc/IMCconfig** as its configuration file.

IMCSERVER

The **IMCSERVER** environment variable specifies the name of a database server entry in the **sqlhosts** file that contains information on connectivity.

The database server can be either local or remote. On UNIX systems that support the C shell, the **IMCSERVER** environment variable can be set by the command.

```
setenv ——— IMCSERVER ——— dbservername ———┤
```

dbservername is the valid name of a database server.

Here *dbservername* must be the name of a database server in the **sqlhosts** file. For more information about **sqlhosts** settings with MaxConnect, see your *Administrator's Guide*. You cannot use MaxConnect unless **IMCSERVER** is set to a valid database server name.

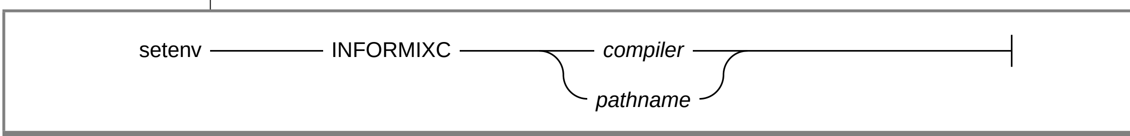
UNIX

INFORMIXC

The **INFORMIXC** environment variable specifies the filename or pathname of the C compiler to be used to compile files that Informix ESQL/C generates. The setting takes effect only during the C compilation stage.

If **INFORMIXC** is not set, the default compiler on most systems is **cc**.

Tip: *On Windows, you pass either -mcc or -bcc options to the esql preprocessor to use either the Microsoft or Borland C compilers.*



compiler is the filename of the C compiler.

pathname is the full pathname of the C compiler.

For example, to specify the GNU C compiler, enter the following command:

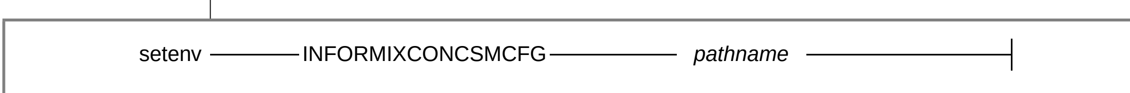
```
setenv INFORMIXC gcc
```

Important: *If you use **gcc**, be aware that Informix assumes that strings are writable, so you need to compile using the **-fwritable-strings** option. Failure to do so can produce unpredictable results, possibly including core dumps.*

IDS

INFORMIXCONCSMCFG

The **INFORMIXCONCSMCFG** environment variable specifies the location of the **concsm.cfg** file that describes communications support modules.



pathname specifies the full pathname of the **concsm.cfg** file.

The following command specifies that the **concsm.cfg** file is in **/usr/myfiles**:

```
setenv INFORMIXCONCSMCFG /usr/myfiles
```

You can also specify a different name for the file. The following example specifies a filename of **csmconfig** in the same directory:

```
setenv INFORMIXCONCSMCFG /usr/myfiles/csmconfig
```

The default location of the **concsm.cfg** file is in **$INFORMIXDIR/etc**. For more information about communications support modules and the contents of the **concsm.cfg** file, refer to the *Administrator's Reference*.

INFORMIXCONRETRY

The **INFORMIXCONRETRY** environment variable specifies the maximum number of *additional* connection attempts that should be made to each database server by the client during the time limit that **INFORMIXCONTIME** specifies.

```
setenv ——— INFORMIXCONRETRY ——— count ———|
```

count is the number of additional attempts to connect to each database server.

For example, the following command sets **INFORMIXCONRETRY** to specify three additional connection attempts (after the initial attempt):

```
setenv INFORMIXCONRETRY 3
```

The default value for **INFORMIXCONRETRY** is one retry after the initial connection attempt. The **INFORMIXCONTIME** setting, described in the following section, takes precedence over the **INFORMIXCONRETRY** setting.

INFORMIXCONTIME

The **INFORMIXCONTIME** environment variable lets you specify that an SQL CONNECT statement should keep trying for at least the given number of seconds before returning an error.

You might encounter connection difficulties related to system or network load problems. For instance, if the database server is busy establishing new SQL client threads, some clients might fail because the database server cannot issue a network function call fast enough. The **INFORMIXCONTIME** and **INFORMIXCONRETRY** variables let you configure your client-side connection capability to retry the connection instead of returning an error.

```
setenv ——— INFORMIXCONTIME ——— value ———|
```

value represents the minimum number of seconds spent in attempts to establish a connection to a database server.

For example, enter the following command to set **INFORMIXCONTIME** to `60` seconds:

```
setenv INFORMIXCONTIME 60
```

If **INFORMIXCONTIME** is set to `60` and **INFORMIXCONRETRY** is set to `3`, attempts to connect to the database server (after the initial attempt at `0` seconds) are made at `20`, `40`, and `60` seconds, if necessary, before aborting. This `20`-second interval is the result of **INFORMIXCONTIME** divided by **INFORMIXCONRETRY**.

If execution of the CONNECT statement involves searching **DBPATH**, the following rules apply:

- All appropriate servers in the **DBPATH** setting are accessed at least once, even though the **INFORMIXCONTIME** value might be exceeded. Thus, the CONNECT statement might take longer than the **INFORMIXCONTIME** time limit to return an error that indicates connection failure or that the database was not found.
- **INFORMIXCONRETRY** specifies how many additional connection attempts should be made for each database server entry in **DBPATH**.

- The **INFORMIXCONTIME** value is divided among the number of database server entries specified in **DBPATH**. Thus, if **DBPATH** contains numerous servers, you should increase the **INFORMIX-CONTIME** value accordingly. For example, if **DBPATH** contains three entries, to spend at least 30 seconds attempting each connection, set **INFORMIXCONTIME** to 90.

The default value for **INFORMIXCONTIME** is 15 seconds. The setting for **INFORMIXCONTIME** takes precedence over the **INFORMIXCONRETRY** setting. Retry efforts could end after the **INFORMIXCONTIME** value is exceeded, but before the **INFORMIXCONRETRY** value is reached.

IDS

INFORMIXCPPMAP

Set the **INFORMIXCPPMAP** environment variable to specify the fully qualified pathname of the map file for C++ programs. Information in the map file includes the database server type, the name of the shared library that supports the database object or value object type, the library entry point for the object, and the C++ library for which an object was built.

```
setenv ——— INFORMIXCPPMAP ——— pathname ———┤
```

pathname	is the directory path where the C++ map file is stored.

The map file is a text file that can have any filename. You can specify several map files, separated by colons (**:**) on UNIX or semicolons (**;**) on Windows.

On UNIX, the default map file is **$INFORMIXDIR/etc/c++map**. On Windows, the default map file is **%INFORMIXDIR%\etc\c++map**.

INFORMIXDIR

The **INFORMIXDIR** environment variable specifies the directory that contains the subdirectories in which your product files are installed. You must always set **INFORMIXDIR**. Verify that **INFORMIXDIR** is set to the full pathname of the directory in which you installed your database server. If you have multiple versions of a database server, set **INFORMIXDIR** to the appropriate directory name for the version that you want to access. For information about when to set **INFORMIXDIR**, see your *Installation Guide*.

```
setenv ──── INFORMIXDIR ──── pathname ────┤
```

pathname is the directory path where the product files are installed.

To set **INFORMIXDIR** to **usr/informix/**, for example, as the installation directory, enter the following command:

```
setenv INFORMIXDIR /usr/informix
```

UNIX

INFORMIXKEYTAB

The **INFORMIXKEYTAB** environment variable specifies the location of the **keytab** file. The **keytab** file contains authentication information that database servers and clients access at connection time, if they use the DCE-GSS communications support module (CSM). It contains key tables that store keys, each of which contains a principal name (database server or user name), type, version, and value.

The database server uses the **keytab** file to find the key to register the database server and to acquire a credential for it. A client application uses the key if the user did not execute **dce_login** with the current operating-system user name (which is the same as the DCE principle name) or did not explicitly provide a credential.

```
setenv ──── INFORMIXKEYTAB ──── pathname
```

pathname specifies the full path of the **keytab** file.

For example, the following command specifies that the name and location of the **keytab** file is **/usr/myfiles/mykeytab**:

```
setenv INFORMIXKEYTAB /usr/myfiles/mykeytab
```

For more information about the DCE-GSS communications support module, see the *Administrator's Guide.*

IDS

INFORMIXOPCACHE

The **INFORMIXOPCACHE** environment variable can specify the size of the memory cache for the staging-area blobspace of the client application.

```
setenv ──── INFORMIXOPCACHE ──── kilobytes
```

kilobytes specifies the value you set for the optical memory cache.

Set the **INFORMIXOPCACHE** environment variable by specifying the size of the memory cache in kilobytes. The specified size must be equal to or smaller than the size of the system-wide configuration parameter, OPCACHEMAX.

If you do not set **INFORMIXOPCACHE**, the default cache size is 128 kilobytes or the size specified in the configuration parameter OPCACHEMAX. The default for OPCACHEMAX is 128 kilobytes. If you set **INFORMIXOPCACHE** to a value of `0`, Optical Subsystem does not use the cache.

INFORMIXSERVER

The **INFORMIXSERVER** environment variable specifies the default database server to which an explicit or implicit connection is made by an SQL API client, the DB-Access utility, or other Informix products. INFORMIXSERVER must be set before you can use Informix client products. It has the following syntax.

```
setenv ──── INFORMIXSERVER ──── dbservername ────┤
```

dbservername is the name of the default database server.

The value of **INFORMIXSERVER** can be a local or remote server, but must correspond to a valid *dbservername* entry in the $INFORMIXDIR**/etc/sqlhosts** file on the computer running the application. The *dbservername* must begin with a lower-case letter and cannot exceed 128 characters. It can include any printable character except the following: uppercase characters, field delimiters (space or tab), newline character, hyphen or minus.

For example, to specify the **coral** database server as the default for connection, enter the following command:

```
setenv INFORMIXSERVER coral
```

INFORMIXSERVER specifies the database server to which an application connects if the CONNECT DEFAULT statement is executed. It also defines the database server to which an initial implicit connection is established if the first statement in an application is not a CONNECT statement.

Important: *You must set **INFORMIXSERVER** even if the application or DB-Access does not use implicit or explicit default connections.*

XPS

For Extended Parallel Server, the **INFORMIXSERVER** environment variable specifies the name of a dbserver group. To specify a coserver name, use the following format:

```
dbservername.coserver_number
```

Here *dbservername* is the value that you assigned to the DBSERVERNAME configuration parameter in the ONCONFIG configuration file, and *coserver_number* is the value that you assigned to the COSERVER configuration parameter for the connection coserver.

Strictly speaking, **INFORMIXSERVER** is not required for initialization. If **INFORMIXSERVER** is not set, however, Extended Parallel Server does not build the **sysmaster** tables. ♦

UNIX

INFORMIXSHMBASE

The **INFORMIXSHMBASE** environment variable affects only client applications connected to Informix databases that use the interprocess communications (IPC) shared-memory (**ipcshm**) protocol.

Important: *Resetting **INFORMIXSHMBASE** requires a thorough understanding of how the application uses memory. Normally you do not reset **INFORMIXSHMBASE**.*

INFORMIXSHMBASE specifies where shared-memory communication segments are attached to the client process so that client applications can avoid collisions with other memory segments that it uses. If you do not set **INFORMIXSHMBASE**, the memory address of the communication segments defaults to an implementation-specific value such as `0x800000`.

setenv —— INFORMIXSHMBASE —— *value* ——|

value is an integer (in kilobytes) used to calculate the memory address.

The database server calculates the memory address where segments are attached by multiplying the value of **INFORMIXSHMBASE** by `1,024`. For example, on a system that uses the C shell, you can set the memory address to the value `0x800000` by entering the following command:

```
setenv INFORMIXSHMBASE 8192
```

For more information, see your *Administrator's Guide* and the *Administrator's Reference*.

INFORMIXSQLHOSTS

The **INFORMIXSQLHOSTS** environment variable specifies where the SQL client or the database server can find connectivity information.

setenv —— INFORMIXSQLHOSTS —— *pathname* ——|

pathname is the full pathname of the connectivity information file.

UNIX

On UNIX systems, the default search path for the file containing connectivity information is **$INFORMIXDIR/etc/sqlhosts**. You can override this default to specify, for example, the **mysqlhosts** file in the /**work**/**envt** directory, by entering the following command:

```
setenv INFORMIXSQLHOSTS /work/envt/mysqlhosts
```

When **INFORMIXSQLHOSTS** is set, the client or database server looks in the specified file for connectivity information. ♦

WIN NT

On Windows, by default, this environment variable points to the computer whose registry contains the SQLHOSTS subkey. For example, to specify that the client or database server look for connectivity information on a computer named **arizona**, enter the following command:

```
set INFORMIXSQLHOSTS = \\arizona
```

♦

For details of the information that **sqlhosts** (or a file with a non-default filename) can provide about connectivity, see your *Administrator's Guide*.

INFORMIXSTACKSIZE

INFORMIXSTACKSIZE environment variable specifies the stack size (in kilobytes) that the database server uses for the primary thread of a client session. You can use **INFORMIXSTACKSIZE** to override the value of the ONCONFIG parameter STACKSIZE for a given application or user.

```
setenv ——— INFORMIXSTACKSIZE ——— size ———|
```

size is an integer, setting the stack size (in kilobytes) for SQL client threads.

For example, to decrease the **INFORMIXSTACKSIZE** to 20 kilobytes, enter the following command:

```
setenv INFORMIXSTACKSIZE 20
```

If **INFORMIXSTACKSIZE** is not set, the stack size is taken from the database server configuration parameter STACKSIZE, or else defaults to a platform-specific value. The default stack size value for the primary thread of an SQL client is 32 kilobytes for nonrecursive database activity.

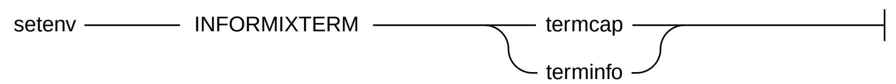
Warning: *For specific instructions on setting this value, see the "Administrator's Reference." If you incorrectly set the value of* ***INFORMIXSTACKSIZE****, it can cause the database server to fail.*

UNIX

INFORMIXTERM

The **INFORMIXTERM** environment variable specifies whether DB-Access should use the information in the **termcap** file or the **terminfo** directory.

On character-based systems, the **termcap** file and **terminfo** directory determine terminal-dependent keyboard and screen capabilities, such as the operation of function keys, color and intensity attributes in screen displays, and the definition of window borders and graphic characters.



If **INFORMIXTERM** is not set, the default setting is **termcap**. When DB-Access is installed on your system, a **termcap** file is placed in the **etc** subdirectory of **$**INFORMIXDIR. This file is a superset of an operating-system **termcap** file.

You can use the **termcap** file that Informix supplies, the system **termcap** file, or a **termcap** file that you create. You must set the **TERMCAP** environment variable if you do not use the default **termcap** file. For information on setting the **TERMCAP** environment variable, see page 3-93.

The **terminfo** directory contains a file for each terminal name that has been defined. The **terminfo** setting for **INFORMIXTERM** is supported only on computers that provide full support for the UNIX System V **terminfo** library. For details, see the machine notes file for your product.

IDS

INF_ROLE_SEP

The **INF_ROLE_SEP** environment variable configures the security feature of role separation when the database server is installed. Role separation enforces separating administrative tasks by people who run and audit the database server. If **INF_ROLE_SEP** is not set, then user **informix** (the default) can perform all administrative tasks.

```
setenv ──────────────── INF_ROLE_SEP ────────────── n ──────┤
```

n is any positive integer.

NT

To enable role separation for database servers on Windows, choose the role-separation option during installation. ♦

If **INF_ROLE_SEP** is set, role separation is implemented and a separate group is specified to serve each of the following responsibilities:

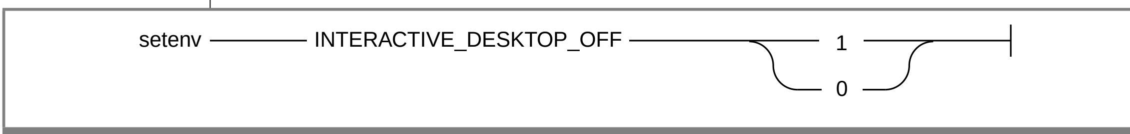
- The database system security officer (DBSSO)
- The audit analysis officer (AAO)
- The standard user

For more information about the security feature of role separation, see the *Trusted Facility Manual*. To learn how to configure role separation when you install your database server, see your *Installation Guide*.

NT

INTERACTIVE_DESKTOP_OFF

The **INTERACTIVE_DESKTOP_OFF** environment variable lets you prevent interaction with the Windows Desktop when an SPL routine executes a SYSTEM( ) command.



1 Prevents the database server from acquiring desktop resources for the user executing the stored procedure

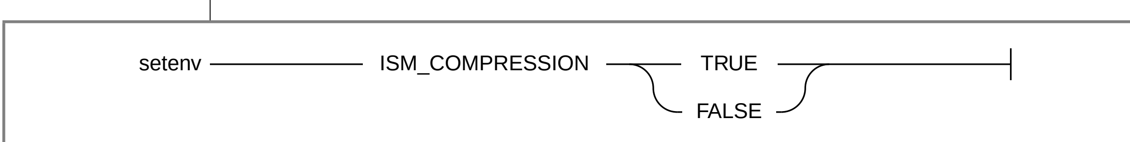
0 SYSTEM( ) commands in a stored procedure can interact with the desktop. This is the default value.

Setting **INTERACTIVE_DESKTOP_OFF** to 1 allows an SPL routine that does not interact with the desktop to execute more quickly. This setting also allows the database server to simultaneously call a greater number of SYSTEM( ) commands because the command no longer depends on a limited operating-system resource (Desktop and WindowStation handles).

If **INTERACTIVE_DESKTOP_OFF** is 1 and an SPL routine attempts to interact with the desktop (for example, with a program such as **notepad.exe** or **cmd.exe**), the routine fails unless the user is a member of the **Administrators** group.

ISM_COMPRESSION

Set the **ISM_COMPRESSION** environment variable in the ON-Bar environment to specify whether the Informix Storage Manager (ISM) should use data compression.



If **ISM_COMPRESSION** is set to TRUE in the environment of the ON-Bar process that makes a request, the ISM server uses a data-compression algorithm to store or retrieve the data specified in that request.

If **ISM_COMPRESSION** is set to FALSE or is not set, the ISM server does not use compression.

ISM_DEBUG_FILE

Set the **ISM_DEBUG_FILE** environment variable in the Informix Storage Manager server environment to specify where to write XBSA messages.

```
setenv ——— ISM_DEBUG_FILE ——— pathname ———|
```

pathname specifies the location of the XBSA message log file.

If you do not set **ISM_DEBUG_FILE**, the XBSA message log is located in the **$INFORMIXDIR/ism/applogs/xbsa.messages** directory on UNIX, or in the **c:\nsr\applogs\xbsa.messages** directory on Windows systems.

ISM_DEBUG_LEVEL

Set the **ISM_DEBUG_LEVEL** environment variable in the ON-Bar environment to control the level of reporting detail recorded in the XBSA messages log. The XBSA shared library writes to this log.

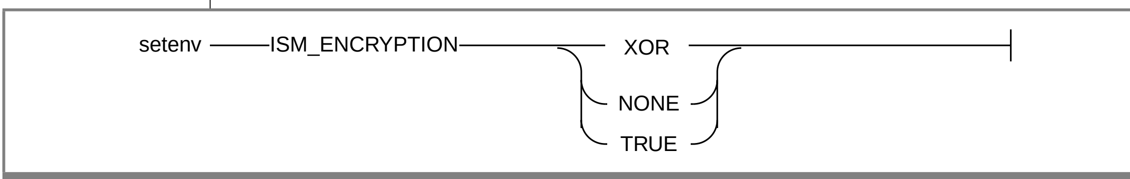
```
setenv ——— ISM_DEBUG_LEVEL ——— value ———|
```

value specifies the level of reporting detail.

You can specify a value between `0` and `9`. If **ISM_DEBUG_LEVEL** is not set, has a NULL value, or has a value outside this range, the default detail level is `1`. A detail level of `0` suppresses all XBSA debugging records. A detail level of `1` reports only XBSA failures.

ISM_ENCRYPTION

Set the **ISM_ENCRYPTION** environment variable in the ON-Bar environment to specify whether Informix Storage Manager (ISM) uses data encryption.



`XOR` uses encryption.

`NONE` does not use encryption.

`TRUE` uses encryption.

If **ISM_ENCRYPTION** is set to NONE or is not set, the ISM server does not use encryption.

If the **ISM_ENCRYPTION** is set to TRUE or XOR in the environment of the ON-Bar process that makes a request, the ISM server uses encryption to store or retrieve the data specified in that request.

ISM_MAXLOGSIZE

Set the **ISM_MAXLOGSIZE** environment variable in the Informix Storage Manager (ISM) server environment to specify the size threshold of the ISM activity log.

size specifies the size threshold (in megabytes) of the activity log.

If **ISM_MAXLOGSIZE** is not set, then the default size limit is 1 megabyte. If **ISM_MAXLOGSIZE** is set to a NULL value, then the threshold is 0 bytes.

ISM_MAXLOGVERS

Set the **ISM_MAXLOGVERS** environment variable in the Informix Storage Manager (ISM) server environment to specify the maximum number of activity-log files to be preserved by the ISM server.

```
setenv ──────── ISM_MAXLOGVERS ──────── value ────────┤
```

value specifies the number of files to be preserved.

If **ISM_MAXLOGVERS** is not set, then the default number of files is four. If the setting is a NULL value, then the ISM server preserves no activity log files.

JAR_TEMP_PATH

Set the **JAR_TEMP_PATH** variable to specify a non-default local file system location where jar management procedures such as **install_jar( )** and **replace_jar( )** can store temporary **.jar** files of the Java virtual machine.

```
setenv ──────── JAR_TEMP_PATH ──────── pathname ────────┤
```

pathname specifies a local directory for temporary **.jar** files.

This directory must have read and write permissions for the user who brings up the database server. If the **JAR_TEMP_PATH** environment variable is not set, temporary copies of **.jar** files are stored in the **/tmp** directory of the local file system for the database server.

JAVA_COMPILER

You can set the **JAVA_COMPILER** environment variable in the Java virtual machine environment to disable JIT compilation.

The NONE and none settings are equivalent. On UNIX systems that support the C shell and on which **JAVA_COMPILER** has been set to NONE or none, you can enable the JIT compiler for the JVM environment by the following command:

```
unset JAVA_COMPILER
```

JVM_MAX_HEAP_SIZE

The **JVM_MAX_HEAP_SIZE** environment variable an set a non-default upper limit on the size of the heap for the Java virtual machine.

size is a positive integer that specifies the maximum size (in megabytes).

For example, the following command sets the maximum heap size at 12Mb:

```
set JVM_MAX_HEAP_SIZE 12
```

If you do not set **JVM_MAX_HEAP_SIZE**, 16MB is the default maximum size.

UNIX

LD_LIBRARY_PATH

The **LD_LIBRARY_PATH** environment variable tells the shell on Solaris systems which directories to search for client or shared Informix general libraries. You must specify the directory that contains your client libraries before you can use the product.

pathname specifies the search path for the library.

The following example sets the **LD_LIBRARY_PATH** environment variable to the desired directory:

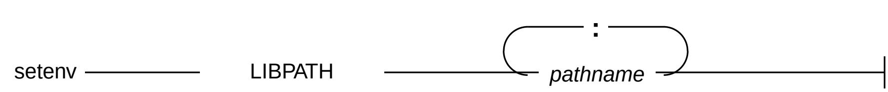
```
setenv LD_LIBRARY_PATH
${INFORMIXDIR}/lib:${INFORMIXDIR}/lib/esql:$LD_LIBRARY_PATH
```

For INTERSOLV DataDirect ODBC Driver on AIX, set **LIBPATH**. For INTERSOLV DataDirect ODBC Driver on HP-UX, set **SHLIB_PATH**.

UNIX

LIBPATH

The **LIBPATH** environment variable tells the shell on AIX systems which directories to search for dynamic-link libraries for the INTERSOLV DataDirect ODBC Driver. You must specify the full pathname for the directory where you installed the product.

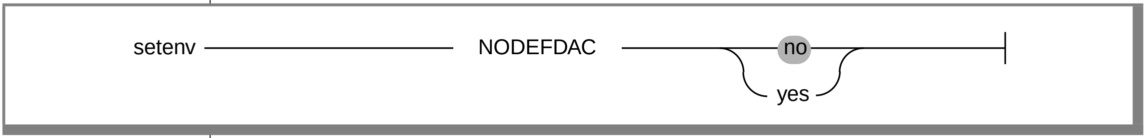


pathname specifies the search path for the libraries.

On Solaris, set **LD_LIBRARY_PATH**. On HP-UX, set **SHLIB_PATH**.

NODEFDAC

When the **NODEFDAC** environment variable is set to yes, it prevents default table privileges (Select, Insert, Update, and Delete) from being granted to PUBLIC when a new table is created in a database that is not ANSI compliant. If you do not set the **NODEFDAC** variable, it is, by default, set to no.



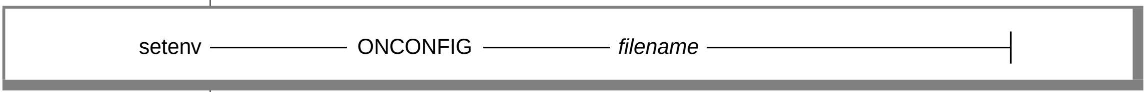
no allows default table privileges to be granted to PUBLIC. Also allows the Execute privilege on a new user-defined routine to be granted to PUBLIC when the user-defined routine is created in Owner mode.

yes prevents default table privileges from being granted to PUBLIC on new tables in a database that is not ANSI compliant. This setting also prevents the Execute privilege for a new user-defined routine from being granted to PUBLIC when the routine is created in Owner mode.

ONCONFIG

The **ONCONFIG** environment variable specifies the name of the active file that holds configuration parameters for the database server. This file is read as input during the initialization procedure. After you prepare the ONCONFIG configuration file, set the **ONCONFIG** environment variable to the name of the file.

If the **ONCONFIG** environment variable is not set, the database server uses configuration values from either the $**ONCONFIG** file or the **$INFORMIXDIR/etc/onconfig** file.



filename is the name of a file in **$INFORMIXDIR/etc** that contains the configuration parameters for your database.

To prepare the ONCONFIG file, make a copy of the **onconfig.std** file and modify the copy. Informix recommends that you name the ONCONFIG file so that it can easily be related to a specific database server. If you have multiple instances of a database server, each instance *must* have its own uniquely named ONCONFIG file.

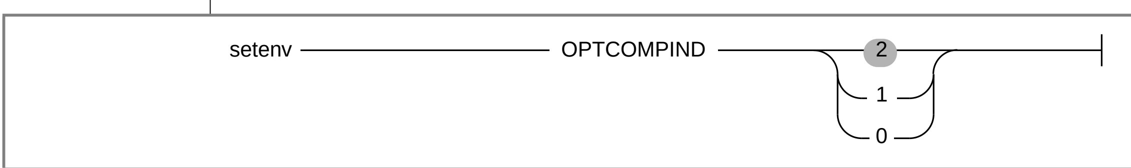
XPS

To prepare the ONCONFIG file for Extended Parallel Server, make a copy of the **onconfig.std** file if you are using a single coserver configuration or make a copy of the **onconfig.xps** file if you are using a multiple coserver configuration. You can use the **onconfig.std** file for a multiple coserver configuration, but you would have to add additional keywords and configuration parameters such as END, NODE, and COSERVER, which are already provided for you in the **onconfig.xps** file. ♦

For more information on configuration parameters and the ONCONFIG file, see the *Administrator's Reference.*

OPTCOMPIND

You can set the **OPTCOMPIND** environment variable so that the optimizer can select the appropriate join method.



0 A nested-loop join is preferred, where possible, over a sort-merge join or a hash join.

1 When the transaction isolation level is *not* Repeatable Read, the optimizer behaves as in setting `2`; otherwise, the optimizer behaves as in setting `0`.

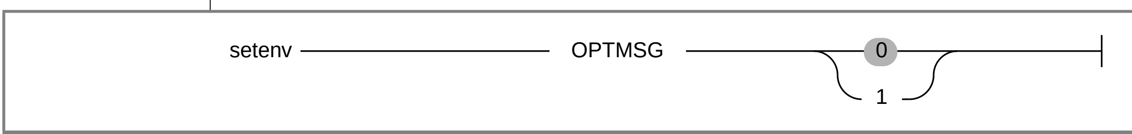
2 Nested-loop joins are not necessarily preferred. The optimizer bases its decision purely on costs, regardless of transaction isolation mode.

When the **OPTCOMPIND** environment variable is not set, the database server uses the value specified for the ONCONFIG configuration parameter OPTCOMPIND. When neither the environment variable nor the configuration parameter is set, the default value is 2.

For more information on the ONCONFIG configuration parameter OPTCOMPIND, see the *Administrator's Reference*. For more information on the different join methods that the optimizer uses, see your *Performance Guide*.

OPTMSG

Set the **OPTMSG** environment variable at runtime before you start an Informix ESQL/C application to enable optimized message transfers (message chaining) for all SQL statements in an application. You also can disable optimized message transfers for statements that require immediate replies, for debugging, or to ensure that the database server processes all messages before the application exits.



0 disables optimized message transfers.

1 enables optimized message transfers and implements the feature for any subsequent connection.

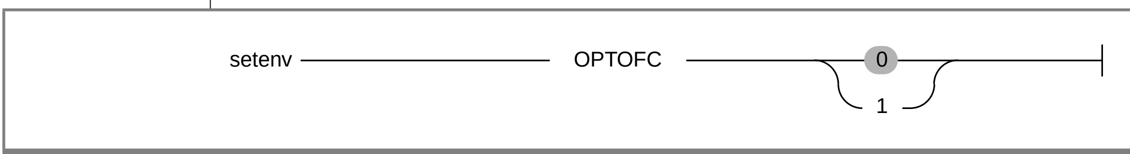
The default value is 0 (zero), which explicitly disables message chaining.

When you set **OPTMSG** within an application, you can activate or deactivate optimized message transfers for each connection or within each thread. To enable optimized message transfers, you must set **OPTMSG** before you establish a connection.

For more information about setting **OPTMSG** and defining related global variables, see the *Informix ESQL/C Programmer's Manual.*

OPTOFC

Set the **OPTOFC** environment variable to enable optimize-OPEN-FETCH-CLOSE functionality in an Informix ESQL/C application that uses DECLARE and OPEN statements to execute a cursor. The **OPTOFC** environment variable reduces the number of message requests between the application and the database server.



0 disables **OPTOFC** for all threads of the application.

1 enables **OPTOFC** for every cursor in every thread of the application.

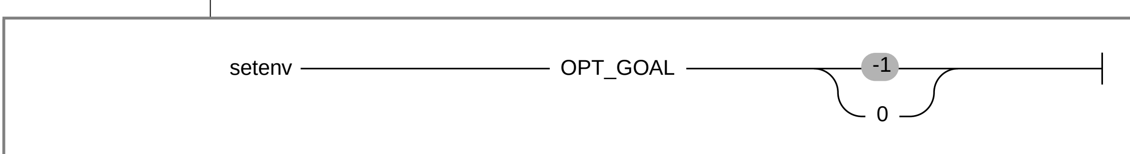
The default value is `0` (zero).

If you set **OPTOFC** from the shell, you must set it before you start the ESQL/C application. For more information about enabling **OPTOFC** and related features, see the *Informix ESQL/C Programmer's Manual*.

UNIX

OPT_GOAL

Set the **OPT_GOAL** environment variable in the user environment, before you start an application, to specify the query performance goal for the optimizer.



0 specifies user-response-time optimization.

-1 specifies total-query-time optimization.

The default behavior is for the optimizer to choose query plans that optimize the total query time.

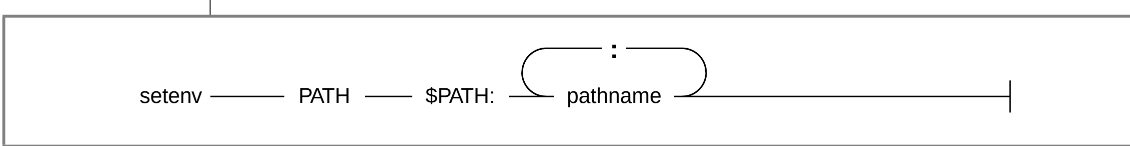
You can also specify the optimization goal for individual queries with optimizer directives or for a session with the SET OPTIMIZATION statement. Both methods take precedence over the **OPT_GOAL** environment variable setting. You can also set the OPT_GOAL configuration parameter for the Dynamic Server system; this method has the lowest level of precedence.

For more information about optimizing queries for your database server, see your *Performance Guide*. For information on the SET OPTIMIZATION statement, see the *Informix Guide to SQL: Syntax*.

PATH

The **PATH** environment variable tells the operating system where to search for executable programs. You must include the directory that contains your Informix product to your **PATH** environment variable before you can use the product. This directory should appear before **$INFORMIXDIR/bin**, which you must also include.

The UNIX **PATH** environment variable tells the shell which directories to search for executable programs. You must add the directory that contains your Informix product to your **PATH** setting before you can use the product.



pathname specifies the search path for the executables.

You can specify the correct search path in various ways. Be sure to include a colon (:) separator between the pathnames on UNIX systems. (Use semicolon (;) as the separator between pathnames on Windows systems.)

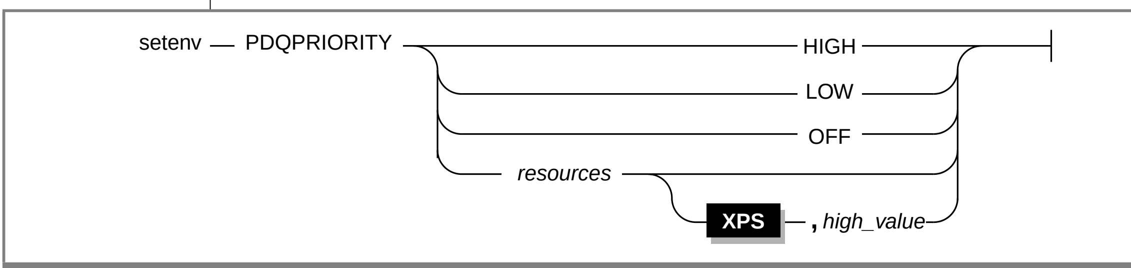
For additional information about how to modify your path, see “Modifying an Environment-Variable Setting” on page 3-10.

PDQPRIORITY

The **PDQPRIORITY** environment variable determines the degree of parallelism that the database server uses and affects how the database server allocates resources, including memory, processors, and disk reads.

XPS

For Extended Parallel Server, the **PDQPRIORITY** environment variable determines only the allocation of memory resources. ♦



resources Is an integer in the range 0 to 100.

Value 0 is the same as OFF (for supported database servers other than Extended Parallel Server only).

Value 1 is the same as LOW.

high_value Optional integer value that requests the maximum percentage of memory (for Extended Parallel Server only). When you specify this value after the *resources* value, you request a range of memory, expressed as a percentage.

Here the HIGH, LOW, and OFF keywords have the following effects:

HIGH When the database server allocates resources among all users, it gives as many resources as possible to the query.

LOW Data is fetched from fragmented tables in parallel.

OFF PDQ processing is turned off (for supported database servers other than Extended Parallel Server).

For supported database servers other than Extended Parallel Server, the *resources* value specifies the query priority level and the amount of resources that the database server uses to process the query. When you specify LOW, the database server uses no forms of parallelism.

IDS

When **PDQPRIORITY** is not set, the default value is `OFF`.

When **PDQPRIORITY** is set to HIGH, the database server determines an appropriate value to use for **PDQPRIORITY** based on several criteria. These include the number of available processors, the fragmentation of tables queried, the complexity of the query, and additional factors. ♦

XPS

The *resources* value establishes the minimum percentage of memory when you also specify *high_value* to request a range of memory allocation. Other parallel operations can occur when the **PDQPRIORITY** setting is LOW.

When the **PDQPRIORITY** environment variable is not set, the default value is the value of the PDQPRIORITY configuration parameter.

When **PDQPRIORITY** is set to `0`, the database server can execute a query in parallel, depending on the number of available processors, the fragmentation of tables queried, the complexity of the query, and so on. **PDQPRIORITY** does not affect the degree of parallelism in Extended Parallel Server. ♦

Usually, the more resources a database server uses, the better its performance for a given query. If the server uses too many resources, however, contention among the resources can result and take resources away from other queries, resulting in degraded performance. For more information on performance considerations for **PDQPRIORITY**, refer to your *Performance Guide*.

An application can override the setting of this environment variable when it issues the SQL statement SET PDQPRIORITY, which the *Informix Guide to SQL: Syntax* describes.

IDS

PLCONFIG

The **PLCONFIG** environment variable specifies the name of the configuration file that the High-Performance Loader (HPL) uses. This file must reside in the **$INFORMIXDIR / etc** directory. If the **PLCONFIG** environment variable is not set, then **$INFORMIXDIR / etc / plconfig** is the default configuration file.

```
setenv ——— PLCONFIG ——— filename ———|
```

filename specifies the simple filename of the configuration file that the High-Performance Loader uses.

For example, to specify the $**INFORMIXDIR/etc / custom.cfg** file as the configuration file for the High-Performance Loader, enter the following command:

```
setenv PLCONFIG custom.cfg
```

For more information, see the *Guide to the High-Performance Loader*.

IDS

PLOAD_LO_PATH

The **PLOAD_LO_PATH** environment variable lets you specify the pathname for smart-large-object handles (which identify the location of smart large objects such as BLOB and CLOB data types).

```
setenv ——— PLOAD_LO_PATH ——— pathname ———|
```

pathname specifies the directory for the smart-large-object handles.

If **PLOAD_LO_PATH** is not set, the default directory is **/tmp**.

For more information, see the *Guide to the High-Performance Loader*.

IDS

PLOAD_SHMBASE

The **PLOAD_SHMBASE** environment variable lets you specify the shared-memory address at which the High-Performance Loader (HPL) **onpload** processes will attach. If **PLOAD_SHMBASE** is not set, the HPL determines which shared-memory address to use.

```
setenv ──── PLOAD_SHMBASE ──── value ────┤
```

value is used to calculate the shared-memory address.

If the **onpload** utility cannot attach, an error appears and you must specify a new value.

The **onpload** utility tries to determine at which address to attach, as follows:

1. Attach at the same address (SHMBASE) as the database server.
2. Attach beyond the database server segments.
3. Attach at the address specified in **PLOAD_SHMBASE**.

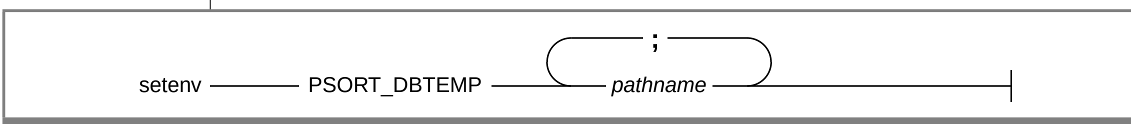
Tip: *Informix recommends that you let the HPL decide where to attach and that you set **PLOAD_SHMBASE** only if necessary to avoid shared-memory collisions between **onpload** and the database server.*

For more information, see the *Guide to the High-Performance Loader*.

PSORT_DBTEMP

The **PSORT_DBTEMP** environment variable specifies a directory or directories where the database server writes the temporary files it uses when it performs a sort.

The database server uses the directory that **PSORT_DBTEMP** specifies, even if the environment variable **PSORT_NPROCS** is not set.



pathname is the name of the UNIX directory used for intermediate writes during a sort.

To set the **PSORT_DBTEMP** environment variable to specify the directory (for example, **/usr/leif/tempsort**), enter the following command:

```
setenv PSORT_DBTEMP /usr/leif/tempsort
```

For maximum performance, specify directories that reside in file systems on different disks.

You might also want to consider setting the environment variable **DBSPACETEMP** to place temporary files used in sorting in dbspaces rather than operating-system files. See the discussion of the **DBSPACETEMP** environment variable in "DBSPACETEMP" on page 3-46.

For additional information about the **PSORT_DBTEMP** environment variable, see your *Administrator's Guide* and your *Performance Guide*.

PSORT_NPROCS

The **PSORT_NPROCS** environment variable enables the database server to improve the performance of the parallel-process sorting package by allocating more threads for sorting.

XPS

PSORT_NPROCS does not necessarily improve sorting speed for Extended Parallel Server because the database server sorts in parallel whether this environment variable is set or not. ♦

Before the sorting package performs a parallel sort, make sure that the database server has enough memory for the sort.

```
setenv ——— PSORT_NPROCS ——— threads ——————————————|
```

threads specifies the maximum number of threads to be used to sort a query. This value cannot be greater than 10.

The following command sets **PSORT_NPROCS** to 4:

```
setenv PSORT_NPROCS 4
```

To disable parallel sorting, enter the following command:

```
unsetenv PSORT_NPROCS
```

Informix recommends that you initially set **PSORT_NPROCS** to 2 when your computer has multiple CPUs. If subsequent CPU activity is lower than I/O activity, you can increase the value of **PSORT_NPROCS**.

Tip: *If the **PDQPRIORITY** environment variable is not set, the database server allocates the minimum amount of memory to sorting. This minimum memory is insufficient to start even two sort threads. If you have not set **PDQPRIORITY**, check the available memory before you perform a large-scale sort (such as an index build) to make sure that you have enough memory.*

Default Values for Ordinary Sorts

If the **PSORT_NPROCS** environment variable is set, the database server uses the specified number of sort threads as an upper limit for ordinary sorts. If **PSORT_NPROCS** is not set, parallel sorting does not take place. The database server uses one thread for the sort. If **PSORT_NPROCS** is set to 0, the database server uses three threads for the sort.

Default Values for Attached Indexes

The default number of threads is different for attached indexes.

If the **PSORT_NPROCS** environment variable is set, you get the specified number of sort threads for each fragment of the index that is being built.

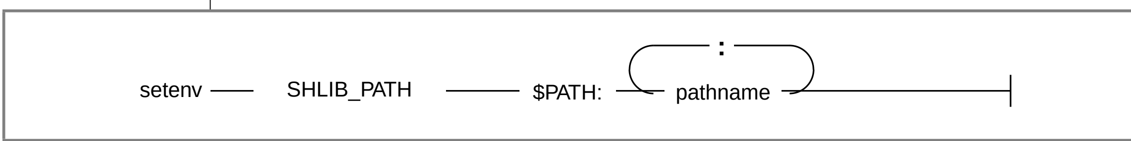
If **PSORT_NPROCS** is not set, or if it is set to `0`, you get two sort threads for each fragment of the index unless you have a single-CPU virtual processor. If you have a single-CPU virtual processor, you get one sort thread for each fragment of the index.

For additional information about the **PSORT_NPROCS** environment variable, see your *Administrator's Guide* and your *Performance Guide*.

UNIX

SHLIB_PATH

The **SHLIB_PATH** environment variable tells the shell on HP-UX systems which directories to search for dynamic-link libraries for the INTERSOLV DataDirect ODBC Driver. You must specify the full pathname for the directory where you installed the product.



pathname specifies the search path for the libraries.

On Solaris systems, set **LD_LIBRARY_PATH**. On AIX systems, set **LIBPATH**.

IDS

STMT_CACHE

Use the **STMT_CACHE** environment variable to control the use of the shared-statement cache on a session. This feature can reduce memory consumption and can speed query processing among different user sessions. Valid **STMT_CACHE** values are `1` and `0`.

1 enables the SQL statement cache.

0 disables the SQL statement cache.

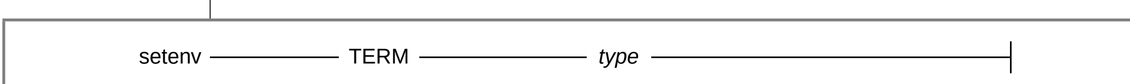
Set the **STMT_CACHE** environment variable for applications that do not use the SET STMT_CACHE statement to control the use of the SQL statement cache. By default, a statement cache of 512 kilobytes is enabled, but this feature can be disabled or set to a non-default size through the STMT_CACHE parameter of the **onconfig.std** file or by the SET STMT_CACHE statement.

This environment variable has no effect if the SQL statement cache is disabled through the configuration file setting. Executing the SET STMT_CACHE statement in the application overrides the **STMT_CACHE** setting.

UNIX

TERM

The **TERM** environment variable is used for terminal handling. It lets DB-Access (and other character-based applications) recognize and communicate with the terminal that you are using.



type specifies the terminal type.

The terminal type specified in the **TERM** setting must correspond to an entry in the **termcap** file or **terminfo** directory. Before you can set the **TERM** environment variable, you must obtain the code for your terminal from the database administrator.

For example, to specify the vt100 terminal, set the **TERM** environment variable by entering the following command:

```
setenv TERM vt100
```

UNIX

TERMCAP

The **TERMCAP** environment variable is used for terminal handling. It tells DB-Access (and other character-based applications) to communicate with the **termcap** file instead of the **terminfo** directory.

setenv — TERMCAP — *pathname* —|

pathname specifies the location of the **termcap** file.

The **termcap** file contains a list of various types of terminals and their characteristics. For example, to provide DB-Access terminal-handling information, which is specified in the **/usr/informix/etc/termcap** file, enter the following command:

```
setenv TERMCAP /usr/informix/etc/termcap
```

You can use set **TERMCAP** in any of the following ways. If several **termcap** files exist, they have the following (descending) order of precedence:

1. The **termcap** file that you create
2. The **termcap** file that Informix supplies (that is, **$INFORMIXDIR/etc/termcap**)
3. The operating-system **termcap** file (that is, **/etc/termcap**)

If you set the **TERMCAP** environment variable, be sure that the **INFORMIXTERM** environment variable is set to the default, **termcap**.

If you do not set the **TERMCAP** environment variable, the system file (that is, **/etc/termcap**) is used by default.

UNIX

TERMINFO

The **TERMINFO** environment variable is used for terminal handling.

The environment variable is supported only on platforms that provide full support for the **terminfo** libraries that System V and Solaris UNIX systems provide.

```
setenv ——— TERMINFO ——— /usr/lib/terminfo ———|
```

TERMINFO tells DB-Access to communicate with the **terminfo** directory instead of the **termcap** file. The **terminfo** directory has subdirectories that contain files that pertain to terminals and their characteristics.

To set **TERMINFO**, enter the following command:

```
setenv TERMINFO /usr/lib/terminfo
```

If you set the **TERMINFO** environment variable, you must also set the **INFORMIXTERM** environment variable to **terminfo**.

UNIX

THREADLIB

Use the **THREADLIB** environment variable to compile multithreaded ESQL/C applications. A multithreaded ESQL/C application lets you establish as many connections to one or more databases as there are threads. These connections can remain active while the application program executes.

The **THREADLIB** environment variable indicates which thread package to use when you compile an application. Currently only the Distributed Computing Environment (DCE) is supported.

```
setenv ——— THREADLIB ——— DCE ———|
```

The **THREADLIB** environment variable is checked when the **-thread** option is passed to the ESQL/C script when you compile a multithreaded ESQL/C application. When you use the **-thread** option while compiling, the ESQL/C script generates an error if **THREADLIB** is not set, or if **THREADLIB** is set to an unsupported thread package.

XPS

XFER_CONFIG

The **XFER_CONFIG** environment variable specifies the location of the **xfer_config** configuration file.

```
setenv ──── XFER_CONFIG ──── pathname ────────┤
```

pathname specifies the location of the **xfer_config** file.

The **xfer_config** file works with the **onxfer** utility to help users migrate from Version 7.x to Version 8.x. It contains various configuration parameter settings that users can modify and a list of tables that users can select to be transferred.

The default **xfer_config** file is located in the **$INFORMIXDIR/etc** directory on UNIX systems or in the **%INFORMIXDIR%\etc** directory in Windows.

Index of Environment Variables

Figure 3-2 on page 3-96 provides an overview of the uses for the various Informix and UNIX environment variables that Version 8.3 and Version 9.2 support. This serves as an index to general topics and lists the related environment variables and the pages where the environment variables are introduced. Where the **Topic** column is empty, the entry refers to the previously listed topic.

GLS

The term *GLS Guide* in the **Page** column in Figure 3-2 indicates environment variables that are described in the *Informix Guide to GLS Functionality*. ♦

Figure 3-2
Uses for Environment Variables

Topic	Environment Variable	Page
Abbreviated year values	**DBCENTURY**	3-28
ANSI/ISO SQL compliance Informix syntax extensions	**DBANSIWARN**	3-27
default table privileges	**NODEFDAC**	3-80
archecker utility	**AC_CONFIG**	3-24
Buffer: fetch size	**FET_BUF_SIZE**	3-55
network size	**IFX_NETBUF_SIZE**	3-60
network pool size	**IFX_NETBUF_PVTPOOL_SIZE**	3-60
BYTE or TEXT data buffer	**DBBLOBBUF**	3-28
Cache: enabling	**STMT_CACHE**	3-92
size for Optical Subsystem	**INFORMIXOPCACHE**	3-68
Client/server: default server	**INFORMIXSERVER**	3-68
shared memory segments	**INFORMIXSHMBASE**	3-69
stacksize for client session	**INFORMIXSTACKSIZE**	3-71
locale of client, server	**CLIENT_LOCALE, DBLOCALE**	GLS Guide
locale for file I/O	**SERVER_LOCALE**	GLS Guide
Code-set conversion code set of client, server	**CLIENT_LOCALE, DB_LOCALE**	GLS Guide
character-string conversion	**DBNLS**	3-40
Communication Support Module: DCE-GSS	**INFORMIXKEYTAB**	3-67
concsm.cfg file	**INFORMIXCONCSMCFG**	3-63
Compiler:	**INFORMIXC**	3-63

(1 of 10)

Topic	Environment Variable	Page
multibyte characters	**CC8BITLEVEL**	GLS Guide
C++	**INFORMIXCPPMAP**	3-66
ESQL/C	**THREADLIB**	3-94
Configuration file: database server	**ONCONFIG**	3-80
ignore variables	**ENVIGNORE**	3-54
Configuration parameter: COSERVER	**INFORMIXSERVER**	3-68
DBSERVERNAME	**INFORMIXSERVER**	3-68
DBSPACETEMP	**DBSPACETEMP**	3-46
DIRECTIVES	**IFX_DIRECTIVES**	3-58
OPCACHEMAX	**INFORMIXOPCACHE**	3-68
OPTCOMPIND	**OPTCOMPIND**	3-81
OPT_GOAL	**OPT_GOAL**	3-83
PDQPRIORITY	**PDQPRIORITY**	3-85
STACKSIZE	**INFORMIXSTACKSIZE**	3-71
Connecting	**INFORMIXCONRETRY**	3-64
	INFORMIXCONTIME	3-65
	INFORMIXSERVER	3-68
	INFORMIXSQLHOSTS	3-70
Data distributions	**DBUPSPACE**	3-52
Database locale	**DB_LOCALE**	GLS Guide
Database server	**INFORMIXSERVER**	3-68
locale for file I/O	**SERVER_LOCALE**	GLS Guide
configuration file	**ONCONFIG**	3-80

(2 of 10)

Topic	Environment Variable	Page
parallel sorting	**PSORT_DBTEMP**	3-89
parallel sorting	**PSORT_NPROCS**	3-89
Database server: restore		3-24
parallelism	**PDQPRIORITY**	3-85
role separation	**INF_ROLE_SEP**	3-73
shared memory	**INFORMIXSHMBASE**	3-69
stacksize	**INFORMIXSTACKSIZE**	3-71
temporary tables	**DBSPACETEMP**	3-46
temporary tables	**DBTEMP**	3-48
temporary tables	**PSORT_DBTEMP**	3-89
Date and time values, formats	**DBCENTURY**	3-28
	DBDATE	3-32; GLS Guide
	DBTIME	3-48; GLS Guide
	GL_DATE, **GL_DATETIME**	GLS Guide
DB-Access utility	**DBACCNOIGN**	3-26
	DBANSIWARN	3-27
	DBDELIMITER	3-35
	DBEDIT	3-35
	DBFLTMASK	3-36
	DBNLS	3-40
	DBPATH	3-42
	DBTEMP	3-48
	FET_BUF_SIZE	3-55

(3 of 10)

Topic	Environment Variable	Page
	INFORMIXSERVER	3-68
	INFORMIXTERM	3-72
	TERM	3-92
	TERMCAP	3-93
	TERMINFO	3-94
dbexport utility	**DBDELIMITER**	3-35
Delimited identifiers	**DELIMIDENT**	3-53
Disk space	**DBUPSPACE**	3-52
Editor	**DBEDIT**	3-35
ESQL/C: ANSI compliance	**DBANSIWARN**	3-27
C compiler	**INFORMIXC**	3-63
DATETIME formatting	**DBTIME**	3-48; GLS Guide
delimited identifiers	**DELIMIDENT**	3-53
multibyte characters	**CLIENT_LOCALE, ESQLMF**	GLS Guide
multithreaded applications	**THREADLIB**	3-94
C preprocessor	**CPFIRST**	3-25
Executable programs	**PATH**	3-84
Fetch buffer size	**FET_BUF_SIZE**	3-55
Filenames: multibyte	**GLS8BITSYS**	GLS Guide
Files: field delimiter	**DBDELIMITER**	3-35
Files: installation	**INFORMIXDIR**	3-67
Files: locale	**CLIENT_LOCALE, DB_LOCALE**	GLS Guide
	SERVER_LOCALE	GLS Guide

(4 of 10)

Topic	Environment Variable	Page
Files: map for C++	**INFORMIXCPPMAP**	3-66
Files: message	**DBLANG**	3-37
Files: temporary	**DBSPACETEMP**	3-46
Files: temporary, for Gateways	**DBTEMP**	3-48
Files: temporary sorting	**PSORT_DBTEMP**	3-89
Files: **termcap**, **terminfo**	**INFORMIXTERM**	3-72
	TERM	3-92
	TERMCAP	3-93
	TERMINFO	3-94
Formats: date and time	**DBDATE**	3-32: GLS Guide
	DBTIME	3-48: GLS Guide
Format: money	**DBMONEY**	3-38 GLS Guide
Gateways	**DBTEMP**	3-48
High-Performance Loader	**DBONPLOAD**	3-41
	PLCONFIG	3-87
	PLOAD_LO_PATH	3-87
	PLOAD_SHMBASE	3-88
Identifiers: delimited	**DELIMIDENT**	3-53
Identifiers: long	**IFX_LONGID**	3-59
Identifiers: multibyte characters	**CLIENT_LOCALE, ESQLMF**	GLS Guide
Informix Storage Manager	**ISM_COMPRESSION**	3-74
	ISM_DEBUG_FILE	3-75

(5 of 10)

Topic	Environment Variable	Page
	ISM_DEBUG_LEVEL	3-75
	ISM_ENCRYPTION	3-76
	ISM_MAXLOGSIZE	3-76
	ISM_MAXLOGVERS	3-77
Installation	**INFORMIXDIR**	3-67
	PATH	3-84
Language environment	**DBLANG**	3-37; GLS Guide
Libraries	**LD_LIBRARY_PATH**	3-79
	LIBPATH	3-79
	SHLIB_PATH	3-91
Locale	**CLIENT_LOCALE, DB_LOCALE**	GLS Guide
	SERVER_LOCALE	GLS Guide
Lock Mode	**IFX_DEF_TABLE_LOCKMODE**	3-56
Long Identifiers	**IFX_LONGID**	3-59
Map file for C++	**INFORMIXCPPMAP**	3-66
Message chaining	**OPTMSG**	3-82
Message files	**DBLANG**	3-37; GLS Guide
Money format	**DBMONEY**	3-38; GLS Guide
Multibyte characters	**CLIENT_LOCALE, DB_LOCALE**	GLS Guide
	SERVER_LOCALE	GLS Guide
Multibyte filter	**ESQLMF**	GLS Guide

(6 of 10)

Topic	Environment Variable	Page
Multithreaded applications	**THREADLIB**	3-94
Network	**DBPATH**	3-42
Nondefault locale	**CLIENT_LOCALE, DB_LOCALE**	GLS Guide
	DBNLS	3-40
	SERVER_LOCALE	GLS Guide
ON-Bar utility	**ISM_COMPRESSION**	3-74
	ISM_DEBUG_LEVEL	3-75
	ISM_ENCRYPTION	3-76
ONCONFIG parameters	See “Configuration parameter”	3-97 to 3-97
Optical Subsystem	**INFORMIXOPCACHE**	3-68
Optimization: directives	**IFX_DIRECTIVES**	3-58
Optimization: message transfers	**OPTMSG**	3-82
Optimization: join method	**OPTCOMPIND**	3-81
Optimization: performance goal	**OPT_GOAL**	3-83
OPTOFC feature	**OPTOFC**	3-83
Parameters	See “Configuration parameter”	3-97 to 3-97
Pathname: **archecker** config file	**AC_CONFIG**	3-24
Pathname: C compiler	**INFORMIXC**	3-63
Pathname: database files	**DBPATH**	3-42
Pathname: executable programs	**PATH**	3-84
Pathname: HPL sblob handles	**PLOAD_LO_PATH**	3-87
Pathname: installation	**INFORMIXDIR**	3-67
Pathname: libraries	**LD_LIBRARY_PATH**	3-79

(7 of 10)

Topic	Environment Variable	Page
	LIBPATH	3-79
	SHLIB_PATH	3-91
Pathname: message files	**DBLANG**	3-37; GLS Guide
Pathname: parallel sorting	**PSORT_DBTEMP**	3-89
Pathname: remote shell	**DBREMOTECMD**	3-45
Pathname: **xfer_config** file	**XFER_CONFIG**	3-95
Printing	**DBPRINT**	3-44
Privileges	**NODEFDAC**	3-73
Query: optimization	**IFX_DIRECTIVES**	3-58
	OPTCOMPIND	3-81
	OPT_GOAL	3-83
Query: prioritization	**PDQPRIORITY**	3-85
Remote shell	**DBREMOTECMD**	3-45
Role separation	**INF_ROLE_SEP**	3-73
Routine: DATETIME formatting	**DBTIME**	3-48; GLS Guide
Server	See "Database server"	3-97
Server locale	**SERVER_LOCALE**	GLS Guide
Shared memory	**INFORMIXSHMBASE**	3-69
	PLOAD_SHMBASE	3-88
Shell: remote	**DBREMOTECMD**	3-45
Shell: search path	**PATH**	3-84
Sorting	**PSORT_DBTEMP**	3-89
	PSORT_NPROCS	3-89

(8 of 10)

Topic	Environment Variable	Page
	DBSPACETEMP	3-46
SQL statement: caching	**STMT_CACHE**	3-92
CONNECT	**INFORMIXCONTIME**	3-65
	INFORMIXSERVER	3-68
CREATE TEMP TABLE	**DBSPACETEMP**	3-46
DESCRIBE FOR UPDATE	**IFX_UPDDESC**	3-61
LOAD, UNLOAD	**DBDELIMITER**	3-35
LOAD, UNLOAD	**DBBLOBBUF**	3-28
SELECT INTO TEMP	**DBSPACETEMP**	3-46
SET PDQPRIORITY	**PDQPRIORITY**	3-85
SET STMT_CACHE	**STMT_CACHE**	3-92
UPDATE STATISTICS	**DBUPSPACE**	3-52
Stacksize	**INFORMIXSTACKSIZE**	3-71
Temporary tables	**DBSPACETEMP**	3-46
	PSORT_DBTEMP	3-89
	DBSPACETEMP	3-46
	DBTEMP	3-48
	PSORT_DBTEMP	3-89
Terminal handling	**INFORMIXTERM**	3-72
	TERM	3-92
	TERMCAP	3-93
	TERMINFO	3-94
Utilities: Archive Checker		3-24

(9 of 10)

Topic	Environment Variable	Page
Utilities: DB-Access	**DBANSIWARN**	3-27
	DBDELIMITER	3-35
	DBEDIT	3-35
	DBFLTMASK	3-36
	DBNLS	3-40
	DBPATH	3-42
	FET_BUF_SIZE	3-55
	INFORMIXSERVER	3-68
	INFORMIXTERM	3-72
	TERM	3-92
	TERMCAP	3-93
	TERMINFO	3-94
Utilities: **dbexport**	**DBDELIMITER**	3-35
Utilities: ON-Bar	**ISM_COMPRESSION**	3-74
	ISM_DEBUG_LEVEL	3-75
	ISM_ENCRYPTION	3-76
Variables: overriding	**ENVIGNORE**	3-54
Year 2000	**DBCENTURY**	3-28

(10 of 10)

Appendix

A

The stores_demo Database

The **stores_demo** database contains a set of tables that describe an imaginary business. The examples in the *Informix Guide to SQL: Syntax*, the *Informix Guide to SQL: Tutorial*, and other Informix manuals are based on this demonstration database. The **stores_demo** database uses the default (U.S. English) locale and is not ANSI compliant.

This appendix contains the following sections:

- The first section describes the structure of the tables in the **stores_demo** database. It identifies the primary key of each table, lists the name and data type of each column, and indicates whether the column has a default value or check constraint. Indexes on columns are also identified and classified as unique allowing duplicate values.
- The second section (“The stores_demo Database Map” on page A-8) shows a graphic map of the tables in the **stores_demo** database and indicates the relationships among columns.
- The third section (“Primary-Foreign Key Relationships” on page A-10) describes the primary-foreign key relationships among columns in tables.
- The final section (“Data in the stores_demo Database” on page A-17) lists the data contained in each table of the **stores_demo** database.

For information on how to create and populate the **stores_demo** database, see the *DB-Access User’s Manual*. For information on how to design and implement a relational database, see the *Informix Guide to Database Design and Implementation*.

Structure of the Tables

The **stores_demo** database contains information about a fictitious sporting-goods distributor that services stores in the western United States. This database includes the following tables:

- **customer** (page A-2)
- **orders** (page A-3)
- **items** (page A-4)
- **stock** (page A-4)
- **catalog** (page A-5)
- **cust_calls** (page A-6)
- **call_type** (page A-7)
- **manufact** (page A-7)
- **state** (page A-8)

Sections that follow describe each table. The unique identifying value for each table (primary key) is shaded and indicated by a key ( ) symbol.

The customer Table

The **customer** table contains information about the retail stores that place orders from the distributor. Figure A-1 shows the columns of the **customer** table.

The **zipcode** column in Figure A-1 is indexed and allows duplicate values.

Figure A-1
The customer Table

Column Name	Data Type	Description
customer_num	SERIAL(101)	System-generated customer number
fname	CHAR(15)	First name of store representative
lname	CHAR(15)	Last name of store representative

(1 of 2)

Column Name	Data Type	Description
company	CHAR(20)	Name of store
address1	CHAR(20)	First line of store address
address2	CHAR(20)	Second line of store address
city	CHAR(15)	City
state	CHAR(2)	State (foreign key to **state** table)
zipcode	CHAR(5)	Zipcode
phone	CHAR(18)	Telephone number

(2 of 2)

The orders Table

The **orders** table contains information about orders placed by the customers of the distributor. Figure A-2 shows the columns of the **orders** table.

Figure A-2
The orders Table

Column Name	Data Type	Description
order_num	SERIAL(1001)	System-generated order number
order_date	DATE	Date order entered
customer_num	INTEGER	Customer number (foreign key to **customer** table)
ship_instruct	CHAR(40)	Special shipping instructions
backlog	CHAR(1)	Indicates order cannot be filled because the item is backlogged: ■ y = yes ■ n = no
po_num	CHAR(10)	Customer purchase order number
ship_date	DATE	Shipping date

(1 of 2)

Column Name	Data Type	Description
ship_weight	DECIMAL(8,2)	Shipping weight
ship_charge	MONEY(6)	Shipping charge
paid_date	DATE	Date order paid

(2 of 2)

The items Table

An order can include one or more items. One row exists in the **items** table for each item in an order. Figure A-3 shows the columns of the **items** table.

Figure A-3
The items Table

Column Name	Data Type	Description
item_num	SMALLINT	Sequentially assigned item number for an order
order_num	INTEGER	Order number (foreign key to **orders** table)
stock_num	SMALLINT	Stock number for item (foreign key to **stock** table)
manu_code	CHAR(3)	Manufacturer code for item ordered (foreign key to **manufact** table)
quantity	SMALLINT	Quantity ordered (value must be > 1)
total_price	MONEY(8)	Quantity ordered * unit price = total price of item

The stock Table

The distributor carries 41 types of sporting goods from various manufacturers. More than one manufacturer can supply an item. For example, the distributor offers racing goggles from two manufacturers and running shoes from six manufacturers.

The **stock** table is a catalog of the items sold by the distributor. Figure A-4 shows the columns of the **stock** table.

Figure A-4
The stock Table

Column Name	Data Type	Description
stock_num	SMALLINT	Stock number that identifies type of item
manu_code	CHAR(3)	Manufacturer code (foreign key to **manufact** table)
description	CHAR(15)	Description of item
unit_price	MONEY(6,2)	Unit price
unit	CHAR(4)	Unit by which item is ordered: ■ Each ■ Pair ■ Case ■ Box
unit_descr	CHAR(15)	Description of unit

The catalog Table

The **catalog** table describes each item in stock. Retail stores use this table when placing orders with the distributor. Figure A-5 shows the columns of the **catalog** table.

Figure A-5
The catalog Table

Column Name	Data Type	Description
catalog_num	SERIAL(10001)	System-generated catalog number
stock_num	SMALLINT	Distributor stock number (foreign key to **stock** table)
manu_code	CHAR(3)	Manufacturer code (foreign key to **manufact** table)

(1 of 2)

Column Name	Data Type	Description
cat_descr	TEXT	Description of item
cat_picture	BYTE	Picture of item (binary data)
cat_advert	VARCHAR(255, 65)	Tag line underneath picture

(2 of 2)

The cust_calls Table

All customer calls for information on orders, shipments, or complaints are logged. The **cust_calls** table contains information about these types of customer calls. Figure A-6 shows the columns of the **cust_calls** table.

Figure A-6
The cust_calls Table

Column Name	Data Type	Description
customer_num	INTEGER	Customer number (foreign key to **customer** table)
call_dtime	DATETIME YEAR TO MINUTE	Date and time when call was received
user_id	CHAR(18)	Name of person logging call (default is user login name)
call_code	CHAR(1)	Type of call (foreign key to **call_type** table)
call_descr	CHAR(240)	Description of call
res_dtime	DATETIME YEAR TO MINUTE	Date and time when call was resolved
res_descr	CHAR(240)	Description of how call was resolved

The call_type Table

The call codes associated with customer calls are stored in the **call_type** table. Figure A-7 shows the columns of the **call_type** table.

Figure A-7
The call_type Table

Column Name	Data Type	Description
call_code	CHAR(1)	Call code
code_descr	CHAR (30)	Description of call type

The manufact Table

Information about the nine manufacturers whose sporting goods are handled by the distributor is stored in the **manufact** table. Figure A-8 shows the columns of the **manufact** table.

Figure A-8
The manufact Table

Column Name	Data Type	Description
manu_code	CHAR(3)	Manufacturer code
manu_name	CHAR(15)	Name of manufacturer
lead_time	INTERVAL DAY(3) TO DAY	Lead time for shipment of orders

The state Table

The **state** table contains the names and postal abbreviations for the 50 states of the United States. Figure A-9 shows the columns of the **state** table.

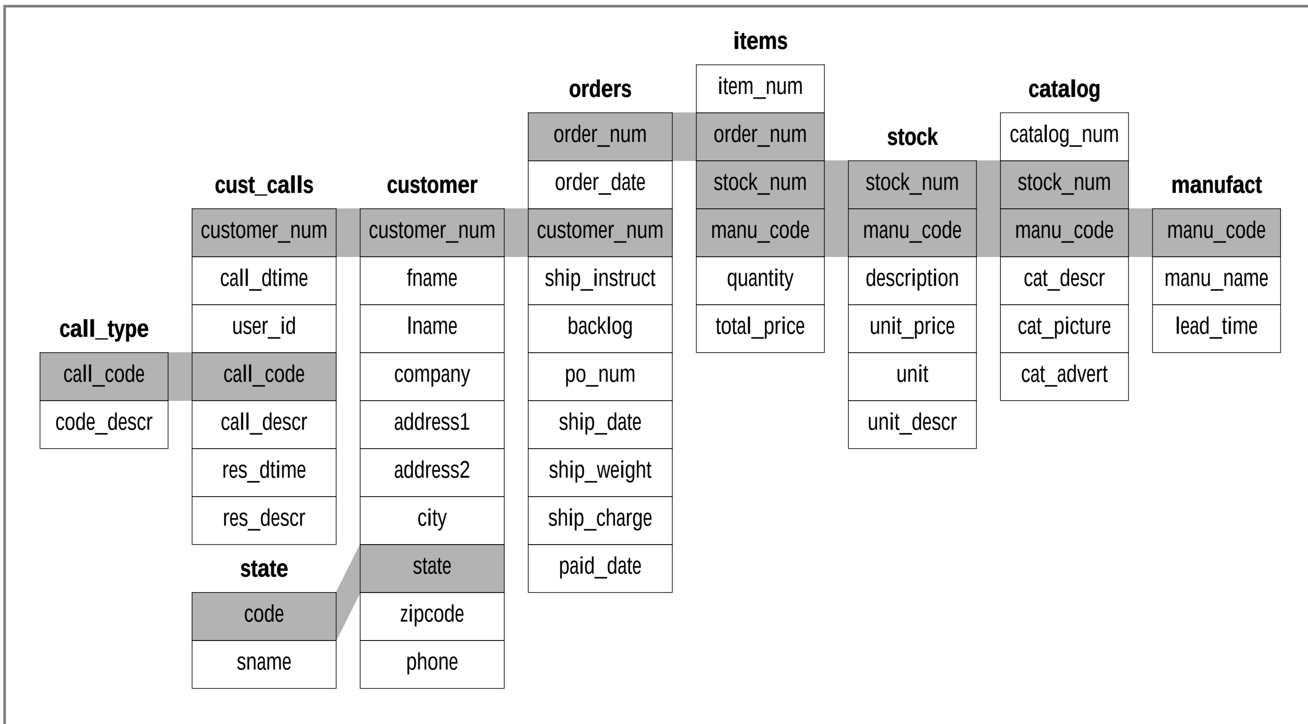
Figure A-9
The state Table

Column Name	Data Type	Description
code	CHAR(2)	State code
sname	CHAR(15)	State name

The stores_demo Database Map

Figure A-10 displays the joins in the **stores_demo** database. The grey shading that connects a column in one table to a column with the same name in another table indicates the relationships, or *joins,* between tables.

Figure A-10
Joins in the stores_demo Database



Primary-Foreign Key Relationships

The tables of the **stores_demo** database are linked by the primary-foreign key relationships that Figure A-10 shows and are identified in this section. This type of relationship is called a *referential constraint* because a foreign key in one table *references* the primary key in another table. Figure A-11 through Figure A-18 show the relationships among tables and how information stored in one table supplements information stored in others.

The customer and orders Tables

The **customer** table contains a **customer_num** column that holds a number that identifies a customer and columns for the customer name, company, address, and telephone number. For example, the row with information about Anthony Higgins contains the number 104 in the **customer_num** column. The **orders** table also contains a **customer_num** column that stores the number of the customer who placed a particular order. In the **orders** table, the **customer_num** column is a foreign key that references the **customer_num** column in the **customer** table. Figure A-11 shows this relationship.

customer Table (detail)

customer_num	fname	lname
101	Ludwig	Pauli
102	Carole	Sadler
103	Philip	Currie
104	Anthony	Higgins

orders Table (detail)

order_num	order_date	customer_num
1001	05/20/1998	104
1002	05/21/1998	101
1003	05/22/1998	104
1004	05/22/1998	106

Figure A-11
Tables That the customer_num Column Joins

According to Figure A-11, customer 104 (Anthony Higgins) has placed two orders, as his customer number appears in two rows of the **orders** table. Because the customer number is a foreign key in the **orders** table, you can retrieve Anthony Higgins's name, address, and information about his orders at the same time.

The orders and items Tables

The **orders** and **items** tables are linked by an **order_num** column that contains an identification number for each order. If an order includes several items, the same order number appears in several rows of the **items** table. In the **items** table, the **order_num** column is a foreign key that references the **order_num** column in the **orders** table. Figure A-12 shows this relationship.

orders Table (detail)

order_num	order_date	customer_num
1001	05/20/1998	104
1002	05/21/1998	101
1003	05/22/1998	104

items Table (detail)

item_num	order_num	stock_num	manu_code
1	1001	1	HRO
4	1002	4	HSK
3	1002	3	HSK
9	1003	9	ANZ
8	1003	8	ANZ
5	1003	5	ANZ

Figure A-12
Tables That the order_num Column Joins

The items and stock Tables

The **items** table and the **stock** table are joined by two columns: the **stock_num** column, which stores a stock number for an item, and the **manu_code** column, which stores a code that identifies the manufacturer. You need both the stock number and the manufacturer code to uniquely identify an item. For example, the item with the stock number `1` and the manufacturer code `HRO` is a Hero baseball glove; the item with the stock number `1` and the manufacturer code `HSK` is a Husky baseball glove.

The same stock number and manufacturer code can appear in more than one row of the **items** table, if the same item belongs to separate orders. In the **items** table, the **stock_num** and **manu_code** columns are foreign keys that reference the **stock_num** and **manu_code** columns in the **stock** table. Figure A-13 shows this relationship.

items Table (detail)

item_num	order_num	stock_num	manu_code
1	1001	1	HRO
1	1002	4	HSK
2	1002	3	HSK
1	1003	9	ANZ
2	1003	8	ANZ
3	1003	5	ANZ
1	1004	1	HRO

stock Table (detail)

stock_num	manu_code	description
1	HRO	baseball gloves
1	HSK	baseball gloves
1	SMT	baseball gloves

Figure A-13
Tables That the stock_num and manu_code Columns Join

The stock and catalog Tables

The **stock** table and **catalog** table are joined by two columns: the **stock_num** column, which stores a stock number for an item, and the **manu_code** column, which stores a code that identifies the manufacturer. You need both columns to uniquely identify an item. In the **catalog** table, the **stock_num** and **manu_code** columns are foreign keys that reference the **stock_num** and **manu_code** columns in the **stock** table. Figure A-14 shows this relationship.

stock Table (detail)

stock_num	manu_code	description
1	HRO	baseball gloves
1	HSK	baseball gloves
1	SMT	baseball gloves

catalog Table (detail)

catalog_num	stock_num	manu_code
10001	1	HRO
10002	1	HSK
10003	1	SMT
10004	2	HRO

Figure A-14
Tables That the stock_num and manu_code Columns Join

The stock and manufact Tables

The **stock** table and the **manufact** table are joined by the **manu_code** column. The same manufacturer code can appear in more than one row of the **stock** table if the manufacturer produces more than one piece of equipment. In the **stock** table, the **manu_code** column is a foreign key that references the **manu_code** column in the **manufact** table. Figure A-15 shows this relationship.

stock Table (detail)

stock_num	manu_code	description
1	HRO	baseball gloves
1	HSK	baseball gloves
1	SMT	baseball gloves

manufact Table (detail)

manu_code	manu_name
NRG	Norge
HSK	Husky
HRO	Hero

Figure A-15
Tables That the manu_code Column Joins

The cust_calls and customer Tables

The **cust_calls** table and the **customer** table are joined by the **customer_num** column. The same customer number can appear in more than one row of the **cust_calls** table if the customer calls the distributor more than once with a problem or question. In the **cust_calls** table, the **customer_num** column is a foreign key that references the **customer_num** column in the **customer** table. Figure A-16 shows this relationship.

customer Table (detail)

customer_num	fname	lname
101	Ludwig	Pauli
102	Carole	Sadler
103	Philip	Currie
104	Anthony	Higgins
105	Raymond	Vector
106	George	Watson

cust_calls Table (detail)

customer_num	call_dtime	user_id
106	1998-06-12 08:20	maryj
127	1998-07-31 14:30	maryj
116	1997-11-28 13:34	mannyh
116	1997-12-21 11:24	mannyh

Figure A-16
Tables That the customer_num Column Joins

The call_type and cust_calls Tables

The **call_type** and **cust_calls** tables are joined by the **call_code** column. The same call code can appear in more than one row of the **cust_calls** table because many customers can have the same *type* of problem. In the **cust_calls** table, the **call_code** column is a foreign key that references the **call_code** column in the **call_type** table. Figure A-17 shows this relationship.

Figure A-17
Tables That the call_code Column Joins

call_type Table (detail)

call_code	**code_descr**
B	billing error
D	damaged goods
I	incorrect merchandise sent
L	late shipment
O	other

cust_calls Table (detail)

customer_num	**call_dtime**	**call_code**
106	1998-06-12 08:20	D
127	1998-07-31 14:30	I
116	1997-11-28 13:34	I
116	1997-12-21 11:24	I

The state and customer Tables

The **state** table and the **customer** table are joined by a column that contains the state code. This column is called **code** in the **state** table and **state** in the **customer** table. If several customers live in the same state, the same state code appears in several rows of the table. In the **customer** table, the **state** column is a foreign key that references the **code** column in the **state** table. Figure A-18 shows this relationship.

customer Table (detail)

customer_num	fname	lname	---	state
101	Ludwig	Pauli	---	CA
102	Carole	Sadler	---	CA
103	Philip	Currie	---	CA

state Table (detail)

code	sname
AK	Alaska
AL	Alabama
AR	Arkansas
AZ	Arizona
CA	California

Figure A-18 *Relationship Between the state Column and the code Column*

Data in the stores_demo Database

The following tables display the data in the **stores_demo** database.

customer Table

customer_num	fname	lname	company	address1	address2	city	state	zipcode	phone
101	Ludwig	Pauli	All Sports Supplies	213 Erstwild Court		Sunnyvale	CA	94086	408-789-8075
102	Carole	Sadler	Sports Spot	785 Geary Street		San Francisco	CA	94117	415-822-1289
103	Philip	Currie	Phil's Sports	654 Poplar	P. O. Box 3498	Palo Alto	CA	94303	650-328-4543
104	Anthony	Higgins	Play Ball!	East Shopping Center	422 Bay Road	Redwood City	CA	94026	650-368-1100
105	Raymond	Vector	Los Altos Sports	1899 La Loma Drive		Los Altos	CA	94022	650-776-3249
106	George	Watson	Watson & Son	1143 Carver Place		Mountain View	CA	94063	650-389-8789
107	Charles	Ream	Athletic Supplies	41 Jordan Avenue		Palo Alto	CA	94304	650-356-9876
108	Donald	Quinn	Quinn's Sports	587 Alvarado		Redwood City	CA	94063	650-544-8729
109	Jane	Miller	Sport Stuff	Mayfair Mart	7345 Ross Blvd.	Sunnyvale	CA	94086	408-723-8789

(1 of 3)

customer_num	fname	lname	company	address1	address2	city	state	zipcode	phone
110	Roy	Jaeger	AA Athletics	520 Topaz Way		Redwood City	CA	94062	650-743-3611
111	Frances	Keyes	Sports Center	3199 Sterling Court		Sunnyvale	CA	94085	408-277-7245
112	Margaret	Lawson	Runners & Others	234 Wyandotte Way		Los Altos	CA	94022	650-887-7235
113	Lana	Beatty	Sportstown	654 Oak Grove		Menlo Park	CA	94025	650-356-9982
114	Frank	Albertson	Sporting Place	947 Waverly Place		Redwood City	CA	94062	650-886-6677
115	Alfred	Grant	Gold Medal Sports	776 Gary Avenue		Menlo Park	CA	94025	650-356-1123
116	Jean	Parmelee	Olympic City	1104 Spinosa Drive		Mountain View	CA	94040	650-534-8822
117	Arnold	Sipes	Kids Korner	850 Lytton Court		Redwood City	CA	94063	650-245-4578
118	Dick	Baxter	Blue Ribbon Sports	5427 College		Oakland	CA	94609	650-655-0011
119	Bob	Shorter	The Triathletes Club	2405 Kings Highway		Cherry Hill	NJ	08002	609-663-6079

(2 of 3)

customer_num	fname	lname	company	address1	address2	city	state	zipcode	phone
120	Fred	Jewell	Century Pro Shop	6627 N. 17th Way		Phoenix	AZ	85016	602-265-8754
121	Jason	Wallack	City Sports	Lake Biltmore Mall	350 W. 23rd Street	Wilmington	DE	19898	302-366-7511
122	Cathy	O'Brian	The Sporting Life	543 Nassau Street		Princeton	NJ	08540	609-342-0054
123	Marvin	Hanlon	Bay Sports	10100 Bay Meadows Road	Suite 1020	Jacksonville	FL	32256	904-823-4239
124	Chris	Putnum	Putnum's Putters	4715 S.E. Adams Blvd	Suite 909C	Bartlesville	OK	74006	918-355-2074
125	James	Henry	Total Fitness Sports	1450 Commonwealth Avenue		Brighton	MA	02135	617-232-4159
126	Eileen	Neelie	Neelie's Discount Sports	2539 South Utica Street		Denver	CO	80219	303-936-7731
127	Kim	Satifer	Big Blue Bike Shop	Blue Island Square	12222 Gregory Street	Blue Island	NY	60406	312-944-5691
128	Frank	Lessor	Phoenix University	Athletic Department	1817 N. Thomas Road	Phoenix	AZ	85008	602-533-1817

(3 of 3)

items Table

item_num	order_num	stock_num	manu_code	quantity	total_price
1	1001	1	HRO	1	250.00
1	1002	4	HSK	1	960.00
2	1002	3	HSK	1	240.00
1	1003	9	ANZ	1	20.00
2	1003	8	ANZ	1	840.00
3	1003	5	ANZ	5	99.00
1	1004	1	HRO	1	250.00
2	1004	2	HRO	1	126.00
3	1004	3	HSK	1	240.00
4	1004	1	HSK	1	800.00
1	1005	5	NRG	10	280.00
2	1005	5	ANZ	10	198.00
3	1005	6	SMT	1	36.00
4	1005	6	ANZ	1	48.00
1	1006	5	SMT	5	125.00
2	1006	5	NRG	5	140.00
3	1006	5	ANZ	5	99.00
4	1006	6	SMT	1	36.00
5	1006	6	ANZ	1	48.00
1	1007	1	HRO	1	250.00
2	1007	2	HRO	1	126.00
3	1007	3	HSK	1	240.00
4	1007	4	HRO	1	480.00

(1 of 3)

item_num	order_num	stock_num	manu_code	quantity	total_price
5	1007	7	HRO	1	600.00
1	1008	8	ANZ	1	840.00
2	1008	9	ANZ	5	100.00
1	1009	1	SMT	1	450.00
1	1010	6	SMT	1	36.00
2	1010	6	ANZ	1	48.00
1	1011	5	ANZ	5	99.00
1	1012	8	ANZ	1	840.00
2	1012	9	ANZ	10	200.00
1	1013	5	ANZ	1	19.80
2	1013	6	SMT	1	36.00
3	1013	6	ANZ	1	48.00
4	1013	9	ANZ	2	40.00
1	1014	4	HSK	1	960.00
2	1014	4	HRO	1	480.00
1	1015	1	SMT	1	450.00
1	1016	101	SHM	2	136.00
2	1016	109	PRC	3	90.00
3	1016	110	HSK	1	308.00
4	1016	114	PRC	1	120.00
1	1017	201	NKL	4	150.00
2	1017	202	KAR	1	230.00
3	1017	301	SHM	2	204.00
1	1018	307	PRC	2	500.00

(2 of 3)

item_num	order_num	stock_num	manu_code	quantity	total_price
2	1018	302	KAR	3	15.00
3	1018	110	PRC	1	236.00
4	1018	5	SMT	4	100.00
5	1018	304	HRO	1	280.00
1	1019	111	SHM	3	1499.97
1	1020	204	KAR	2	90.00
2	1020	301	KAR	4	348.00
1	1021	201	NKL	2	75.00
2	1021	201	ANZ	3	225.00
3	1021	202	KAR	3	690.00
4	1021	205	ANZ	2	624.00
1	1022	309	HRO	1	40.00
2	1022	303	PRC	2	96.00
3	1022	6	ANZ	2	96.00
1	1023	103	PRC	2	40.00
2	1023	104	PRC	2	116.00
3	1023	105	SHM	1	80.00
4	1023	110	SHM	1	228.00
5	1023	304	ANZ	1	170.00
6	1023	306	SHM	1	190.00

(3 of 3)

call_type Table

call_code	code_descr
B	billing error
D	damaged goods
I	incorrect merchandise sent
L	late shipment
O	other

orders Table

order_num	order_date	customer_num	ship_instruct	backlog	po_num	ship_date	ship_weight	ship_charge	paid_date
1001	05/20/1998	104	express	n	B77836	06/01/1998	20.40	10.00	07/22/1998
1002	05/21/1998	101	PO on box; deliver back door only	n	9270	05/26/1998	50.60	15.30	06/03/1998
1003	05/22/1998	104	express	n	B77890	05/23/1998	35.60	10.80	06/14/1998
1004	05/22/1998	106	ring bell twice	y	8006	05/30/1998	95.80	19.20	
1005	05/24/1998	116	call before delivery	n	2865	06/09/1998	80.80	16.20	06/21/1998
1006	05/30/1998	112	after 10AM	y	Q13557		70.80	14.20	
1007	05/31/1998	117		n	278693	06/05/1998	125.90	25.20	
1008	06/07/1998	110	closed Monday	y	LZ230	07/06/1998	45.60	13.80	07/21/1998
1009	06/14/1998	111	door next to grocery	n	4745	06/21/1998	20.40	10.00	08/21/1998
1010	06/17/1998	115	deliver 776 King St. if no answer	n	429Q	06/29/1998	40.60	12.30	08/22/1998
1011	06/18/1998	104	express	n	B77897	07/03/1998	10.40	5.00	08/29/1998
1012	06/18/1998	117		n	278701	06/29/1998	70.80	14.20	

(1 of 2)

order_num	order_date	customer_num	ship_instruct	backlog	po_num	ship_date	ship_weight	ship_charge	paid_date
1013	06/22/1998	104	express	n	B77930	07/10/1998	60.80	12.20	07/31/1998
1014	06/25/1998	106	ring bell, kick door loudly	n	8052	07/03/1998	40.60	12.30	07/10/1998
1015	06/27/1998	110	closed Mondays	n	MA003	07/16/1998	20.60	6.30	08/31/1998
1016	06/29/1998	119	delivery entrance off Camp St.	n	PC6782	07/12/1998	35.00	11.80	
1017	07/09/1998	120	North side of clubhouse	n	DM354331	07/13/1998	60.00	18.00	
1018	07/10/1998	121	SW corner of Biltmore Mall	n	S22942	07/13/1998	70.50	20.00	08/06/1998
1019	07/11/1998	122	closed til noon Mondays	n	Z55709	07/16/1998	90.00	23.00	08/06/1998
1020	07/11/1998	123	express	n	W2286	07/16/1998	14.00	8.50	09/20/1998
1021	07/23/1998	124	ask for Elaine	n	C3288	07/25/1998	40.00	12.00	08/22/1998
1022	07/24/1998	126	express	n	W9925	07/30/1998	15.00	13.00	09/02/1998
1023	07/24/1998	127	no deliveries after 3 p.m.	n	KF2961	07/30/1998	60.00	18.00	08/22/1998

(2 of 2)

stock Table

stock_num	manu_code	description	unit_price	unit	unit_descr
1	HRO	baseball gloves	250.00	case	10 gloves/case
1	HSK	baseball gloves	800.00	case	10 gloves/case
1	SMT	baseball gloves	450.00	case	10 gloves/case
2	HRO	baseball	126.00	case	24/case
3	HSK	baseball bat	240.00	case	12/case
3	SHM	baseball bat	280.00	case	12/case
4	HSK	football	960.00	case	24/case
4	HRO	football	480.00	case	24/case
5	NRG	tennis racquet	28.00	each	each
5	SMT	tennis racquet	25.00	each	each
5	ANZ	tennis racquet	19.80	each	each
6	SMT	tennis ball	36.00	case	24 cans/case
6	ANZ	tennis ball	48.00	case	24 cans/case
7	HRO	basketball	600.00	case	24/case
8	ANZ	volleyball	840.00	case	24/case
9	ANZ	volleyball net	20.00	each	each
101	PRC	bicycle tires	88.00	box	4/box
101	SHM	bicycle tires	68.00	box	4/box
102	SHM	bicycle brakes	220.00	case	4 sets/case
102	PRC	bicycle brakes	480.00	case	4 sets/case
103	PRC	front derailleur	20.00	each	each
104	PRC	rear derailleur	58.00	each	each
105	PRC	bicycle wheels	53.00	pair	pair
105	SHM	bicycle wheels	80.00	pair	pair

(1 of 3)

stock_num	manu_code	description	unit_price	unit	unit_descr
106	PRC	bicycle stem	23.00	each	each
107	PRC	bicycle saddle	70.00	pair	pair
108	SHM	crankset	45.00	each	each
109	PRC	pedal binding	30.00	case	6 pairs/case
109	SHM	pedal binding	200.00	case	4 pairs/case
110	PRC	helmet	236.00	case	4/case
110	ANZ	helmet	244.00	case	4/case
110	SHM	helmet	228.00	case	4/case
110	HRO	helmet	260.00	case	4/case
110	HSK	helmet	308.00	case	4/case
111	SHM	10-spd, assmbld	499.99	each	each
112	SHM	12-spd, assmbld	549.00	each	each
113	SHM	18-spd, assmbld	685.90	each	each
114	PRC	bicycle gloves	120.00	case	10 pairs/case
201	NKL	golf shoes	37.50	each	each
201	ANZ	golf shoes	75.00	each	each
201	KAR	golf shoes	90.00	each	each
202	NKL	metal woods	174.00	case	2 sets/case
202	KAR	std woods	230.00	case	2 sets/case
203	NKL	irons/wedges	670.00	case	2 sets/case
204	KAR	putter	45.00	each	each
205	NKL	3 golf balls	312.00	case	24/case
205	ANZ	3 golf balls	312.00	case	24/case
205	HRO	3 golf balls	312.00	case	24/case
301	NKL	running shoes	97.00	each	each

(2 of 3)

stock_num	manu_code	description	unit_price	unit	unit_descr
301	HRO	running shoes	42.50	each	each
301	SHM	running shoes	102.00	each	each
301	PRC	running shoes	75.00	each	each
301	KAR	running shoes	87.00	each	each
301	ANZ	running shoes	95.00	each	each
302	HRO	ice pack	4.50	each	each
302	KAR	ice pack	5.00	each	each
303	PRC	socks	48.00	box	24 pairs / box
303	KAR	socks	36.00	box	24 pair / box
304	ANZ	watch	170.00	box	10 / box
304	HRO	watch	280.00	box	10 / box
305	HRO	first-aid kit	48.00	case	4 / case
306	PRC	tandem adapter	160.00	each	each
306	SHM	tandem adapter	190.00	each	each
307	PRC	infant jogger	250.00	each	each
308	PRC	twin jogger	280.00	each	each
309	HRO	ear drops	40.00	case	20 / case
309	SHM	ear drops	40.00	case	20 / case
310	SHM	kick board	80.00	case	10 / case
310	ANZ	kick board	89.00	case	12 / case
311	SHM	water gloves	48.00	box	4 pairs / box
312	SHM	racer goggles	96.00	box	12 / box
312	HRO	racer goggles	72.00	box	12 / box
313	SHM	swim cap	72.00	box	12 / box
313	ANZ	swim cap	60.00	box	12 / box

(3 of 3)

catalog Table

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
10001	1	HRO	Brown leather. Specify first baseman's or infield / outfield style. Specify right- or left-handed.	<BYTE value>	Your First Season's Baseball Glove
10002	1	HSK	Babe Ruth signature glove. Black leather. Infield / outfield style. Specify right- or left-handed.	<BYTE value>	All-Leather, Hand-Stitched, Deep-Pockets, Sturdy Webbing that Won't Let Go
10003	1	SMT	Catcher's mitt. Brown leather. Specify right- or left-handed.	<BYTE value>	A Sturdy Catcher's Mitt With the Perfect Pocket
10004	2	HRO	Jackie Robinson signature glove. Highest Professional quality, used by National League.	<BYTE value>	Highest Quality Ball Available, from the Hand-Stitching to the Robinson Signature
10005	3	HSK	Pro-style wood. Available in sizes: 31, 32, 33, 34, 35.	<BYTE value>	High-Technology Design Expands the Sweet Spot
10006	3	SHM	Aluminum. Blue with black tape. 31", 20 oz or 22 oz; 32", 21 oz or 23 oz; 33", 22 oz or 24 oz.	<BYTE value>	Durable Aluminum for High School and Collegiate Athletes
10007	4	HSK	Norm Van Brocklin signature style.	<BYTE value>	Quality Pigskin with Norm Van Brocklin Signature
10008	4	HRO	NFL-Style pigskin.	<BYTE value>	Highest Quality Football for High School and Collegiate Competitions

(1 of 13)

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
10009	5	NRG	Graphite frame. Synthetic strings.	<BYTE value>	Wide Body Amplifies Your Natural Abilities by Providing More Power Through Aerodynamic Design
10010	5	SMT	Aluminum frame. Synthetic strings.	<BYTE value>	Mid-Sized Racquet for the Improving Player
10011	5	ANZ	Wood frame, cat-gut strings.	<BYTE value>	Antique Replica of Classic Wooden Racquet Built with Cat-Gut Strings
10012	6	SMT	Soft yellow color for easy visibility in sunlight or artificial light.	<BYTE value>	High-Visibility Tennis, Day or Night
10013	6	ANZ	Pro-core. Available in neon yellow, green, and pink.	<BYTE value>	Durable Construction Coupled with the Brightest Colors Available
10014	7	HRO	Indoor. Classic NBA style. Brown leather.	<BYTE value>	Long-Life Basketballs for Indoor Gymnasiums
10015	8	ANZ	Indoor. Finest leather. Professional quality.	<BYTE value>	Professional Volleyballs for Indoor Competitions
10016	9	ANZ	Steel eyelets. Nylon cording. Double-stitched. Sanctioned by the National Athletic Congress.	<BYTE value>	Sanctioned Volleyball Netting for Indoor Professional and Collegiate Competition

(2 of 13)

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
10017	101	PRC	Reinforced, hand-finished tubular. Polyurethane belted. Effective against punctures. Mixed tread for super wear and road grip.	<BYTE value>	Ultimate in Puncture Protection, Tires Designed for In-City Riding
10018	101	SHM	Durable nylon casing with butyl tube for superior air retention. Center-ribbed tread with herringbone side. Coated sidewalls resist abrasion.	<BYTE value>	The Perfect Tire for Club Rides or Training
10019	102	SHM	Thrust bearing and coated pivot washer/ spring sleeve for smooth action. Slotted levers with soft gum hoods. Two-tone paint treatment. Set includes calipers, levers, and cables.	<BYTE value>	Thrust-Bearing and Spring-Sleeve Brake Set Guarantees Smooth Action
10020	102	PRC	Computer-aided design with low-profile pads. Cold-forged alloy calipers and beefy caliper bushing. Aero levers. Set includes calipers, levers, and cables.	<BYTE value>	Computer Design Delivers Rigid Yet Vibration-Free Brakes
10021	103	PRC	Compact leading-action design enhances shifting. Deep cage for super-small granny gears. Extra strong construction to resist off-road abuse.	<BYTE value>	Climb Any Mountain: ProCycle's Front Derailleur Adds Finesse to Your ATB

(3 of 13)

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
10022	104	PRC	Floating trapezoid geometry with extra thick parallelogram arms. 100-tooth capacity. Optimum alignment with any freewheel.	<BYTE value>	Computer-Aided Design Engineers 100-Tooth Capacity Into ProCycle’s Rear Derailleur
10023	105	PRC	Front wheels laced with 15g spokes in a 3-cross pattern. Rear wheels laced with 14g spikes in a 3-cross pattern.	<BYTE value>	Durable Training Wheels That Hold True Under Toughest Conditions
10024	105	SHM	Polished alloy. Sealed-bearing, quick-release hubs. Double-butted. Front wheels are laced 15g / 2-cross. Rear wheels are laced 15g / 3-cross.	<BYTE value>	Extra Lightweight Wheels for Training or High-Performance Touring
10025	106	PRC	Hard anodized alloy with pearl finish. 6mm hex bolt hardware. Available in lengths of 90-140mm in 10mm increments.	<BYTE value>	ProCycle Stem with Pearl Finish
10026	107	PRC	Available in three styles: Men’s racing; Men’s touring; and Women’s. Anatomical gel construction with lycra cover. Black or black / hot pink.	<BYTE value>	The Ultimate In Riding Comfort, Lightweight With Anatomical Support

(4 of 13)

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
10027	108	SHM	Double or triple crankset with choice of chainrings. For double crankset, chainrings from 38-54 teeth. For triple crankset, chainrings from 24-48 teeth.	<BYTE value>	Customize Your Mountain Bike With Extra-Durable Crankset
10028	109	PRC	Steel toe clips with nylon strap. Extra wide at buckle to reduce pressure.	<BYTE value>	Classic Toeclip Improved to Prevent Soreness at Clip Buckle
10029	109	SHM	Ingenious new design combines button on sole of shoe with slot on a pedal plate to give riders new options in riding efficiency. Choose full or partial locking. Four plates mean both top and bottom of pedals are slotted—no fishing around when you want to engage full power. Fast unlocking ensures safety when maneuverability is paramount.	<BYTE value>	Ingenious Pedal/Clip Design Delivers Maximum Power and Fast Unlocking
10030	110	PRC	Super-lightweight. Meets both ANSI and Snell standards for impact protection. 7.5 oz. Quick-release shadow buckle.	<BYTE value>	Feather-Light, Quick-Release, Maximum Protection Helmet

(5 of 13)

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
10031	110	ANZ	No buckle so no plastic touches your chin. Meets both ANSI and Snell standards for impact protection. 7.5 oz. Lycra cover.	<BYTE value>	Minimum Chin Contact, Feather-Light, Maximum Protection Helmet
10032	110	SHM	Dense outer layer combines with softer inner layer to eliminate the mesh cover, no snagging on brush. Meets both ANSI and Snell standards for impact protection. 8.0 oz.	<BYTE value>	Mountain Bike Helmet: Smooth Cover Eliminates the Worry of Brush Snags But Delivers Maximum Protection
10033	110	HRO	Newest ultralight helmet uses plastic shell. Largest ventilation channels of any helmet on the market. 8.5 oz.	<BYTE value>	Lightweight Plastic with Vents Assures Cool Comfort Without Sacrificing Protection
10034	110	HSK	Aerodynamic (teardrop) helmet covered with anti-drag fabric. Credited with shaving 2 seconds/mile from winner's time in Tour de France time-trial. 7.5 oz.	<BYTE value>	Teardrop Design Used by Yellow Jerseys, You Can Time the Difference
10035	111	SHM	Light-action shifting 10 speed. Designed for the city commuter with shock-absorbing front fork and drilled eyelets for carry-all racks or bicycle trailers. Internal wiring for generator lights. 33 lbs.	<BYTE value>	Fully Equipped Bicycle Designed for the Serious Commuter Who Mixes Business With Pleasure

(6 of 13)

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
10036	112	SHM	Created for the beginner enthusiast. Ideal for club rides and light touring. Sophisticated triple-butted frame construction. Precise index shifting. 28 lbs.	<BYTE value>	We Selected the Ideal Combination of Touring Bike Equipment, then Turned It Into This Package Deal: High-Performance on the Roads, Maximum Pleasure Everywhere
10037	113	SHM	Ultra-lightweight. Racing frame geometry built for aerodynamic handlebars. Cantilever brakes. Index shifting. High-performance gearing. Quick-release hubs. Disk wheels. Bladed spokes.	<BYTE value>	Designed for the Serious Competitor, The Complete Racing Machine
10038	114	PRC	Padded leather palm and stretch mesh merged with terry back; Available in tan, black, and cream. Sizes S, M, L, XL.	<BYTE value>	Riding Gloves for Comfort and Protection
10039	201	NKL	Designed for comfort and stability. Available in white & blue or white & brown. Specify size.	<BYTE value>	Full-Comfort, Long-Wearing Golf Shoes for Men and Women
10040	201	ANZ	Guaranteed waterproof. Full leather upper. Available in white, bone, brown, green, and blue. Specify size.	<BYTE value>	Waterproof Protection Ensures Maximum Comfort and Durability In All Climates

(7 of 13)

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
10041	201	KAR	Leather and leather mesh for maximum ventilation. Waterproof lining to keep feet dry. Available in white and gray or white and ivory. Specify size.	<BYTE value>	Karsten's Top Quality Shoe Combines Leather and Leather Mesh
10042	202	NKL	Complete starter set utilizes gold shafts. Balanced for power.	<BYTE value>	Starter Set of Woods, Ideal for High School and Collegiate Classes
10043	202	KAR	Full set of woods designed for precision control and power performance.	<BYTE value>	High-Quality Woods Appropriate for High School Competitions or Serious Amateurs
10044	203	NKL	Set of eight irons includes 3 through 9 irons and pitching wedge. Originally priced at $489.00.	<BYTE value>	Set of Irons Available From Factory at Tremendous Savings: Discontinued Line
10045	204	KAR	Ideally balanced for optimum control. Nylon-covered shaft.	<BYTE value>	High-Quality Beginning Set of Irons Appropriate for High School Competitions
10046	205	NKL	Fluorescent yellow.	<BYTE value>	Long Drive Golf Balls: Fluorescent Yellow
10047	205	ANZ	White only.	<BYTE value>	Long Drive Golf Balls: White

(8 of 13)

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
10048	205	HRO	Combination fluorescent yellow and standard white.	<BYTE value>	HiFlier Golf Balls: Case Includes Fluorescent Yellow and Standard White
10049	301	NKL	Super shock-absorbing gel pads disperse vertical energy into a horizontal plane for extraordinary cushioned comfort. Great motion control. Men’s only. Specify size.	<BYTE value>	Maximum Protection For High-Mileage Runners
10050	301	HRO	Engineered for serious training with exceptional stability. Fabulous shock absorption. Great durability. Specify men’s / women’s, size.	<BYTE value>	Pronators and Supinators Take Heart: A Serious Training Shoe For Runners Who Need Motion Control
10051	301	SHM	For runners who log heavy miles and need a durable, supportive, stable platform. Mesh / synthetic upper gives excellent moisture dissipation. Stability system uses rear antipronation platform and forefoot control plate for extended protection during high-intensity training. Specify men’s / women’s size.	<BYTE value>	The Training Shoe Engineered for Marathoners and Ultra-Distance Runners

(9 of 13)

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
10052	301	PRC	Supportive, stable racing flat. Plenty of forefoot cushioning with added motion control. Women's only. D widths available. Specify size.	<BYTE value>	A Woman's Racing Flat That Combines Extra Forefoot Protection With a Slender Heel
10053	301	KAR	Anatomical last holds your foot firmly in place. Feather-weight cushioning delivers the responsiveness of a racing flat. Specify men's / women's size.	<BYTE value>	Durable Training Flat That Can Carry You Through Marathon Miles
10054	301	ANZ	Cantilever sole provides shock absorption and energy rebound. Positive traction shoe with ample toe box. Ideal for runners who need a wide shoe. Available in men's and women's. Specify size.	<BYTE value>	Motion Control, Protection, and Extra Toebox Room
10055	302	KAR	Reusable ice pack with velcro strap. For general use. Velcro strap allows easy application to arms or legs.	<BYTE value>	Finally, an Ice Pack for Achilles Injuries and Shin Splints That You Can Take to the Office
10056	303	PRC	Neon nylon. Perfect for running or aerobics. Indicate color: Fluorescent pink, yellow, green, and orange.	<BYTE value>	Knock Their Socks Off With YOUR Socks

(10 of 13)

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
10057	303	KAR	100% nylon blend for optimal wicking and comfort. We've taken out the cotton to eliminate the risk of blisters and reduce the opportunity for infection. Specify men's or women's.	<BYTE value>	100% Nylon Blend Socks - No Cotton
10058	304	ANZ	Provides time, date, dual display of lap/cumulative splits, 4-lap memory, 10 hr count-down timer, event timer, alarm, hour chime, waterproof to 50m, velcro band.	<BYTE value>	Athletic Watch w/4-Lap Memory
10059	304	HRO	Split timer, waterproof to 50m. Indicate color: Hot pink, mint green, space black.	<BYTE value>	Waterproof Triathlete Watch In Competition Colors
10060	305	HRO	Contains ace bandage, anti-bacterial cream, alcohol cleansing pads, adhesive bandages of assorted sizes, and instant-cold pack.	<BYTE value>	Comprehensive First-Aid Kit Essential for Team Practices, Team Traveling
10061	306	PRC	Converts a standard tandem bike into an adult/child bike. User-tested assembly instructions	<BYTE value>	Enjoy Bicycling With Your Child on a Tandem; Make Your Family Outing Safer

(11 of 13)

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
10062	306	SHM	Converts a standard tandem bike into an adult/child bike. Lightweight model.	<BYTE value>	Consider a Touring Vacation for the Entire Family: A Lightweight, Touring Tandem for Parent and Child
10063	307	PRC	Allows mom or dad to take the baby out too. Fits children up to 21 pounds. Navy blue with black trim.	<BYTE value>	Infant Jogger Keeps A Running Family Together
10064	308	PRC	Allows mom or dad to take both children! Rated for children up to 18 pounds.	<BYTE value>	As Your Family Grows, Infant Jogger Grows With You
10065	309	HRO	Prevents swimmer's ear.	<BYTE value>	Swimmers Can Prevent Ear Infection All Season Long
10066	309	SHM	Extra-gentle formula. Can be used every day for prevention or treatment of swimmer's ear.	<BYTE value>	Swimmer's Ear Drops Specially Formulated for Children
10067	310	SHM	Blue heavy-duty foam board with Shimara or team logo.	<BYTE value>	Exceptionally Durable, Compact Kickboard for Team Practice
10068	310	ANZ	White. Standard size.	<BYTE value>	High-Quality Kickboard
10069	311	SHM	Swim gloves. Webbing between fingers promotes strengthening of arms. Cannot be used in competition.	<BYTE value>	Hot Training Tool - Webbed Swim Gloves Build Arm Strength and Endurance

(12 of 13)

catalog_num	stock_num	manu_code	cat_descr	cat_picture	cat_advert
10070	312	SHM	Hydrodynamic egg-shaped lens. Ground-in anti-fog elements; Available in blue or smoke.	<BYTE value>	Anti-Fog Swimmer's Goggles: Quantity Discount
10071	312	HRO	Durable competition-style goggles. Available in blue, grey, or white.	<BYTE value>	Swim Goggles: Traditional Rounded Lens For Greater Comfort
10072	313	SHM	Silicone swim cap. One size. Available in white, silver, or navy. Team Logo Imprinting Available.	<BYTE value>	Team Logo Silicone Swim Cap
10073	314	ANZ	Silicone swim cap. Squared-off top. One size. White	<BYTE value>	Durable Squared-off Silicone Swim Cap
10074	315	HRO	Re-usable ice pack. Store in the freezer for instant first-aid. Extra capacity to accommodate water and ice.	<BYTE value>	Water Compartment Combines With Ice to Provide Optimal Orthopedic Treatment

(13 of 13)

cust_calls Table

customer_num	call_dtime	user_id	call_code	call_descr	res_dtime	res_descr
106	1998-06-12 8:20	maryj	D	Order was received, but two of the cans of ANZ tennis balls within the case were empty.	1998-06-12 8:25	Authorized credit for two cans to customer, issued apology. Called ANZ buyer to report the QA problem.
110	1998-07-07 10:24	richc	L	Order placed one month ago (6/7) not received.	1998-07-07 10:30	Checked with shipping (Ed Smith). Order sent yesterday- we were waiting for goods from ANZ. Next time will call with delay if necessary.
119	1998-07-01 15:00	richc	B	Bill does not reflect credit from previous order.	1998-07-02 8:21	Spoke with Jane Akant in Finance. She found the error and is sending new bill to customer.
121	1998-07-10 14:05	maryj	O	Customer likes our merchandise. Requests that we stock more types of infant joggers. Will call back to place order.	1998-07-10 14:06	Sent note to marketing group of interest in infant joggers.

(1 of 2)

customer_num	call_dtime	user_id	call_code	call_descr	res_dtime	res_descr
127	1998-07-31 14:30	maryj	I	Received Hero watches (item # 304) instead of ANZ watches.		Sent memo to shipping to send ANZ item 304 to customer and pickup HRO watches. Should be done tomorrow, 8/1.
116	1997-11-28 13:34	mannyn	I	Received plain white swim caps (313 ANZ) instead of navy with team logo (313 SHM).	1997-11-28 16:47	Shipping found correct case in warehouse and express mailed it in time for swim meet.
116	1997-12-21 11:24	mannyn	I	Second complaint from this customer! Received two cases right-handed outfielder gloves (1 HRO) instead of one case lefties.	1997-12-27 08:19	Memo to shipping (Ava Brown) to send case of left-handed gloves, pick up wrong case; memo to billing requesting 5% discount to placate customer due to second offense and lateness of resolution because of holiday.

(2 of 2)

manufact Table

manu_code	manu_name	lead_time
ANZ	Anza	5
HSK	Husky	5
HRO	Hero	4
NRG	Norge	7
SMT	Smith	3
SHM	Shimara	30
KAR	Karsten	21
NKL	Nikolus	8
PRC	ProCycle	9

state Table

code	sname	code	sname
AK	Alaska	MT	Montana
AL	Alabama	NE	Nebraska
AR	Arkansas	NC	North Carolina
AZ	Arizona	ND	North Dakota
CA	California	NH	New Hampshire
CT	Connecticut	NJ	New Jersey
CO	Colorado	NM	New Mexico
DC	Washington, D.C.	NV	Nevada
DE	Delaware	NY	New York
FL	Florida	OH	Ohio

(1 of 2)

code	sname	code	sname
GA	Georgia	OK	Oklahoma
HI	Hawaii	OR	Oregon
IA	Iowa	PA	Pennsylvania
ID	Idaho	PR	Puerto Rico
IL	Illinois	RI	Rhode Island
IN	Indiana	SC	South Carolina
KY	Kentucky	TN	Tennessee
LA	Louisiana	TX	Texas
MA	Massachusetts	UT	Utah
MD	Maryland	VA	Virginia
ME	Maine	VT	Vermont
MI	Michigan	WA	Washington
MN	Minnesota	WI	Wisconsin
MO	Missouri	WV	West Virginia
MS	Mississippi	WY	Wyoming

(2 of 2)

The sales_demo and superstores_demo Databases

In addition to the **stores_demo** database that is described in detail in Appendix A, Informix products include the following demonstration databases:

XPS

- The **sales_demo** database illustrates a dimensional schema for data-warehousing applications. ♦

IDS

- The **superstores_demo** database illustrates an object-relational schema. ♦

This appendix discusses the structures of these two demonstration databases.

For information on how to create and populate the demonstration databases, including relevant SQL files, see the *DB-Access User's Manual*. For conceptual information about demonstration databases, see the *Informix Guide to Database Design and Implementation*.

XPS

The sales_demo Database

Your database server product contains SQL scripts for the **sales_demo** dimensional database. The **sales_demo** database provides an example of a simple data-warehousing environment and works in conjunction with the **stores_demo** database. The scripts for the **sales_demo** database create new tables and add extra rows to the **items** and **orders** tables of **stores_demo**.

To create the **sales_demo** database, you must first create the **stores_demo** database with the logging option. Once you create the **stores_demo** database, you can execute the scripts that create and load the **sales_demo** database from DB-Access. The files are named **createdw.sql** and **loaddw.sql**.

Dimensional Model of the sales_demo Database

Figure B-1 gives an overview of the tables in the **sales_demo** database.

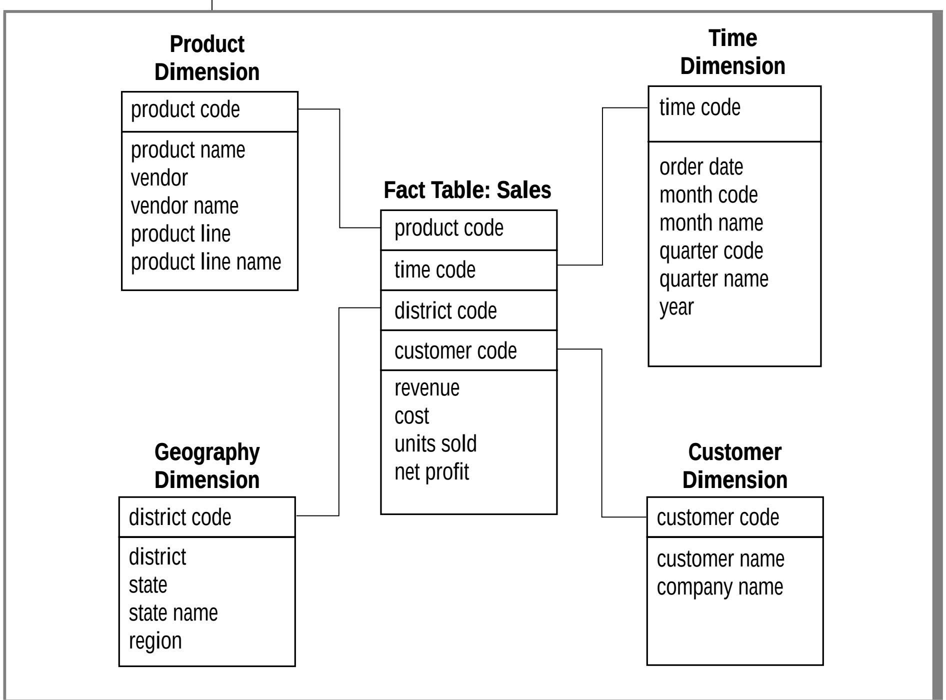


Figure B-1
The sales_demo Dimensional Data Model

For information on how to create and populate the **sales_demo** database, see the *DB-Access User's Manual*. For information on how to design and implement dimensional databases, see the *Informix Guide to Database Design and Implementation*. For information on the **stores_demo** database, see Appendix A.

Structure of the sales_demo Tables

The **sales_demo** database includes the following tables:

- **customer**
- **geography**
- **product**
- **sales**
- **time**

The tables are listed alphabetically, not in the order in which they are created. The **customer**, **geography**, **product**, and **time** tables are the dimensions for the **sales** fact table.

The **sales_demo** database is not ANSI compliant.

The following sections describe the column names, data types, and column descriptions for each table. A SERIAL field serves as the primary key for the **district_code** column of the **geography** table. However, the primary and foreign key relationships that exist between the fact (**sales**) table and its dimension tables are not defined because data-loading performance improves dramatically when the database server does not enforce constraint checking.

The customer Table

The **customer** table contains information about sales customers. Figure B-2 shows the columns of the **customer** table.

Figure B-2
The customer Table

Name	Type	Description
customer_code	INTEGER	Customer code
customer_name	CHAR(31)	Customer name
company_name	CHAR(20)	Company name

The geography Table

The **geography** table contains information about the sales district and region. Figure B-3 shows the columns of the **geography** table.

Figure B-3
The geography Table

Name	Type	Description
district_code	SERIAL	District code
district_name	CHAR(15)	District name
state_code	CHAR(2)	State code
state_name	CHAR(18)	State name
region	SMALLINT	Region name

The product Table

The **product** table contains information about the products sold through the data warehouse. Figure B-4 shows the columns of the **product** table.

Figure B-4
The product Table

Name	Type	Description
product_code	INTEGER	Product code
product_name	CHAR(31)	Product name
vendor_code	CHAR(3)	Vendor code
vendor_name	CHAR(15)	Vendor name
product_line_code	SMALLINT	Product line code
product_line_name	CHAR(15)	Name of product line

The sales Table

The **sales** fact table contains information about product sales and has a pointer to each dimension table. For example, the **customer_code** column references the **customer** table, the **district_code** column references the **geography** table, and so on. The **sales** table also contains the measures for the units sold, revenue, cost, and net profit. Figure B-5 shows the columns of the **sales** table.

Figure B-5
The sales Table

Name	Type	Description
customer_code	INTEGER	Customer code (references **customer**)
district_code	SMALLINT	District code (references **geography**)
time_code	INTEGER	Time code (references **time**)
product_code	INTEGER	Product code (references **product**)
units_sold	SMALLINT	Number of units sold
revenue	MONEY(8,2)	Amount of sales revenue
cost	MONEY(8,2)	Cost of sale
net_profit	MONEY(8,2)	Net profit of sale

The time Table

The **time** table contains time information about the sale. Figure B-6 shows the columns of the **time** table.

Figure B-6
The time Table

Name	Type	Description
time_code	INTEGER	Time code
order_date	DATE	Order date
month_code	SMALLINT	Month code
month_name	CHAR(10)	Name of month
quarter_code	SMALLINT	Quarter code
quarter_name	CHAR(10)	Name of quarter
year	INTEGER	Year

IDS

The superstores_demo Database

SQL files and user-defined routines (UDRs) that are provided with DB-Access let you derive the **superstores_demo** object-relational database.

The **superstores_demo** database uses the default locale and is not ANSI compliant.

This section provides the following **superstores_demo** information:

- The structure of all the tables in the **superstores_demo** database
- A list and definition of the extended data types that **superstores_demo** uses
- A map of table hierarchies
- The primary-foreign key relationships among the columns in the database tables

For information on how to create and populate the **superstores_demo** database, see the *DB-Access User's Manual*. For information on how to work with object-relational databases, see the *Informix Guide to Database Design and Implementation*. For information on the **stores_demo** database on which **superstores_demo** is based, see Appendix A.

Structure of the superstores_demo Tables

The **superstores_demo** database includes the following tables. Although many tables have the same name as **stores_demo** tables, they are different. The tables are listed alphabetically, not in the order in which they are created.

- **call_type**
- **catalog**
- **cust_calls**
- **customer**
  - **retail_customer** (*new*)
  - **whlsale_customer** (*new*)
- **items**
- **location** (*new*)
  - **location_non_us** (*new*)
  - **location_us** (*new*)
- **manufact**
- **orders**
- **region** (*new*)
- **sales_rep** (*new*)
- **state**
- **stock**
- **stock_discount** (*new*)
- **units** (*new*)

This section lists the names, data types, and descriptions of the columns for each table in the **superstores_demo** database. The unique identifying value for each table (primary key) is shaded and indicated by a key ( ) symbol. Columns that represent extended data types are discussed in "User-Defined Routines and Extended Data Types" on page B-22. Primary-foreign key relationships between the tables are outlined in "Referential Relationships" on page B-25.

The call_type Table

The call codes associated with customer calls are stored in the **call_type** table. Figure B-7 shows the columns of the **call_type** table.

Figure B-7
The call_type Table

Name	Type	Description
call_code	CHAR(1)	Call code
codel_descr	CHAR (30)	Description of call code

The catalog Table

The **catalog** table describes each item in stock. Retail stores use this table when placing orders with the distributor. Figure B-8 shows the columns of the **catalog** table.

Figure B-8
The catalog Table

Name	Type	Description
catalog_num	SERIAL(1001)	System-generated catalog number
stock_num	SMALLINT	Distributor stock number (foreign key to **stock** table)
manu_code	CHAR(3)	Manufacturer code (foreign key to **stock** table)

(1 of 2)

Name	Type	Description
unit	CHAR(4)	Unit by which item is ordered (foreign key to **stock** table)
advert	ROW (picture BLOB, caption LVARCHAR)	Picture of item and caption
advert_descr	CLOB	Tag line underneath picture

(2 of 2)

The cust_calls Table

All customer calls for information on orders, shipments, or complaints are logged. The **cust_calls** table contains information about these types of customer calls. Figure B-9 shows the columns of the **cust_calls** table.

Figure B-9
The cust_calls Table

Name	Type	Description
customer_num	INTEGER	Customer number (foreign key to **customer** table)
call_dtime	DATETIME YEAR TO MINUTE	Date and time call received
user_id	CHAR(18)	Name of person logging call (default is user login name)
call_code	CHAR(1)	Type of call (foreign key to **call_type** table)
call_descr	CHAR(240)	Description of call
res_dtime	DATETIME YEAR TO MINUTE	Date and time call resolved
res_descr	CHAR(240)	Description of how call was resolved

The customer, retail_customer, and whlsale_customer Tables

In this hierarchy, **retail_customer** and **whlsale_customer** are sub-tables that are created under the **customer** supertable, as Figure B-25 on page B-25 shows.

For information about table hierarchies, see the *Informix Guide to Database Design and Implementation*.

The customer Table

The **customer** table contains information about the retail stores that place orders from the distributor. Figure B-10 shows the columns of the **customer** table.

Figure B-10
The customer Table

Name	Type	Description
customer_num	SERIAL	Unique customer identifier
customer_type	CHAR(1)	Code to indicate type of customer: ■ R = retail ■ W = wholesale
customer_name	name_t	Name of customer
customer_loc	INTEGER	Location of customer (foreign key to **location** table)
contact_dates	LIST(DATETIME YEAR TO DAY NOT NULL)	Dates of contact with customer
cust_discount	percent	Customer discount

(1 of 2)

Name	Type	Description
credit_status	CHAR(1)	Customer credit status: ■ D = deadbeat ■ L = lost ■ N = new ■ P = preferred ■ R = regular

(2 of 2)

The retail_customer Table

The **retail_customer** table contains general information about retail customers. Figure B-11 shows the columns of the **retail_customer** table.

Figure B-11
The retail_customer Table

Name	Type	Description
customer_num	SERIAL	Unique customer identifier
customer_type	CHAR(1)	Code to indicate type of customer: ■ R = retail ■ W = wholesale
customer_name	name_t	Name of customer
customer_loc	INTEGER	Location of customer
contact_dates	LIST(DATETIME YEAR TO DAY NOT NULL)	Dates of contact with customer
cust_discount	percent	Customer discount

(1 of 2)

Name	Type	Description
credit_status	CHAR(1)	Customer credit status: ■ D = deadbeat ■ L = lost ■ N = new ■ P = preferred ■ R = regular
credit_num	CHAR(19)	Credit card number
expiration	DATE	Expiration data of credit card

(2 of 2)

The whlsale_customer Table

The **whlsale_customer** table contains general information about wholesale customers. Figure B-12 shows the columns of the **whlsale_customer** table.

Figure B-12
The whlsale_customer Table

Name	Type	Description
customer_num	SERIAL	Unique customer identifier
customer_type	CHAR(1)	Code to indicate type of customer: ■ R = retail ■ W = wholesale
customer_name	name_t	Name of customer
customer_loc	INTEGER	Location of customer
contact_dates	LIST(DATETIME YEAR TO DAY NOT NULL)	Dates of contact with customer
cust_discount	percent	Customer discount

(1 of 2)

Name	Type	Description
credit_status	CHAR(1)	Customer credit status: ■ D = deadbeat ■ L = lost ■ N = new ■ P = preferred ■ R = regular
resale_license	CHAR(15)	Resale license number
terms_net	SMALLINT	Net term in days

(2 of 2)

The items Table

An order can include one or more items. One row exists in the **items** table for each item in an order. Figure B-13 shows the columns of the **items** table.

Figure B-13
The items Table

Name	Type	Description
item_num	SMALLINT	Sequentially assigned item number for an order
order_num	INT8	Order number (foreign key to **orders** table)
stock_num	SMALLINT	Stock number for item (foreign key to **stock** table)
manu_code	CHAR(3)	Manufacturer code for item ordered (foreign key to **stock** table)

(1 of 2)

Name	Type	Description
unit	CHAR(4)	Unit by which item is ordered (foreign key to **stock** table)
quantity	SMALLINT	Quantity ordered (value must be > 1)
item_subtotal	MONEY(8,2)	Quantity ordered * unit price = total price of item

(2 of 2)

The location, location_non_us, and location_us Tables

In this hierarchy, **location_non_us** and **location_us** are subtables that are created under the **location** supertable, as shown in the diagram in "Table Hierarchies" on page B-25. For information about table hierarchies, see the *Informix Guide to Database Design and Implementation*.

The location Table

The **location** table contains general information about the locations (addresses) that the database tracks. Figure B-14 shows the columns of the **location** table.

Figure B-14
The location Table

Name	Type	Description
location_id	SERIAL	Unique identifier for location
loc_type	CHAR(2)	Code to indicate type of location
company	VARCHAR(20)	Name of company
street_addr	LIST(VARCHAR(25) NOT NULL)	Street address
city	VARCHAR(25)	City for address
country	VARCHAR(25)	Country for address

The location_non_us Table

The **location_non_us** table contains specific address information for locations (addresses) that are outside the United States. Figure B-15 shows the columns of the **location_non_us** table.

Figure B-15
The location_non_us Table

Name	Type	Description
location_id	SERIAL	Unique identifier for location
loc_type	CHAR(2)	Code to indicate type of location
company	VARCHAR(20)	Name of company
street_addr	LIST(VARCHAR(25) NOT NULL)	Street address
city	VARCHAR(25)	City for address
country	VARCHAR(25)	Country for address
province_code	CHAR(2)	Province code
zipcode	CHAR(9)	Zip code
phone	CHAR(15)	Phone number

The location_us Table

The **location_us** table contains specific address information for locations (addresses) that are in the United States. Figure B-16 shows the columns of the **location_us** table.

Figure B-16
The location_us Table

Name	Type	Description
location_id	SERIAL	Unique identifier for location
loc_type	CHAR(2)	Code to indicate type of location
company	VARCHAR(20)	Name of company
street_addr	LIST(VARCHAR(25) NOT NULL)	Street address
city	VARCHAR(25)	City for address
country	VARCHAR(25)	Country for address
state_code	CHAR(2)	State code (foreign key to **state** table)
zip	CHAR(9)	Zip code
phone	CHAR(15)	Phone number

The manufact Table

Information about the manufacturers whose sporting goods are handled by the distributor is stored in the **manufact** table. Figure B-17 shows the columns of the **manufact** table.

Figure B-17
The manufact Table

Name	Type	Description
manu_code	CHAR(3)	Manufacturer code
manu_name	VARCHAR(15)	Name of manufacturer
lead_time	INTERVAL DAY(3) TO DAY	Lead time for shipment of orders
manu_loc	INTEGER	Manufacturer location (foreign key to **location** table)
manu_account	CHAR(32)	Distributor account number with manufacturer
account_status	CHAR(1)	Status of account with manufacturer
terms_net	SMALLINT	Distributor terms with manufacturer (in days)
discount	percent	Distributor volume discount with manufacturer

The orders Table

The **orders** table contains information about orders placed by the customers of the distributor. Figure B-18 shows the columns of the **orders** table.

Figure B-18
The orders Table

Name	Type	Description
order_num	SERIAL8(1001)	System-generated order number
order_date	DATE	Date order entered
customer_num	INTEGER	Customer number (foreign key to **customer** table)
shipping	ship_t	Special shipping instructions
backlog	BOOLEAN	Indicates order cannot be filled because the item is back ordered
po_num	CHAR(10)	Customer purchase order number
paid_date	DATE	Date order paid

The region Table

The **region** table contains information about the sales regions for the distributor. Figure B-19 shows the columns of the **region** table.

Figure B-19
The region Table

Name	Type	Description
region_num	SERIAL	System-generated region number
region_name	VARCHAR(20) UNIQUE	Name of sales region
region_loc	INTEGER	Location of region office (foreign key to **location** table)

The sales_rep Table

The **sales_rep** table contains information about the sales representatives for the distributor. Figure B-20 shows the columns of the **sales_rep** table.

Figure B-20
The sales_rep Table

Name	Type	Description
rep_num	SERIAL(101)	System-generated sales rep number
name	name_t	Name of sales rep
region_num	INTEGER	Region in which sales rep works (foreign key to the **region** table)
home_office	BOOLEAN	Home office location of sales rep
sales	SET(ROW (month DATETIME YEAR TO MONTH, amount MONEY) NOT NULL)	Amount of monthly sales for rep
commission	percent	Commission rate for sales rep

The state Table

The **state** table contains the names and postal abbreviations for the 50 states of the United States as well as sales tax information. Figure B-21 shows the columns of the **state** table.

Figure B-21
The state Table

Name	Type	Description
code	CHAR(2)	State code
sname	CHAR(15)	State name
sales_tax	percent	State sales tax

The stock Table

The **stock** table is a catalog of the items sold by the distributor. Figure B-22 shows the columns of the **stock** table.

Figure B-22
The stock Table

Name	Type	Description
stock_num	SMALLINT	Stock number that identifies type of item
manu_code	CHAR(3)	Manufacturer code (foreign key to **manufact**)
unit	CHAR(4)	Unit by which item is ordered
description	VARCHAR(15)	Description of item
unit_price	MONEY(6,2)	Unit price
min_reord_qty	SMALLINT	Minimum reorder quantity
min_inv_qty	SMALLINT	Quantity of stock below which item should be reordered
manu_item_num	CHAR(20)	Manufacturer item number
unit_cost	MONEY(6,2)	Distributor cost per unit of item from manufacturer
status	CHAR(1)	Status of item: ■ A = active ■ D = discontinued ■ N = no order
bin_num	INTEGER	Bin number
qty_on_hand	SMALLINT	Quantity in stock
bigger_unit	CHAR(4)	Stock unit for next larger unit (for same stock_num and manu_code)
per_bigger_unit	SMALLINT	How many of this item in bigger_unit

The stock_discount Table

The **stock_discount** table contains information about stock discounts. (There is no primary key). Figure B-23 shows the columns of the **stock_discount** table.

Figure B-23
The stock_discount Table

Name	Type	Description
discount_id	SERIAL	System-generated discount identifier
stock_num	SMALLINT	Distributor stock number (part of foreign key to **stock** table)
manu_code	CHAR(3)	Manufacturer code (part of foreign key to **stock** table)
unit	CHAR(4)	Unit by which item is ordered (each, pair, case, and so on) (foreign key to **units** table; part of foreign key to **stock** table)
unit_discount	percent	Unit discount during sale period
start_date	DATE	Discount start date
end_date	DATE	Discount end date

The units Table

The **units** table contains information about the units in which the inventory items can be ordered. Each item in the **stock** table is available in one or more types of container. Figure B-24 shows the columns of the **units** table.

Figure B-24
The units Table

Name	Type	Description
unit_name	CHAR(4)	Units by which an item is ordered (each, pair, case, box)
unit_descr	VARCHAR(15)	Description of units

User-Defined Routines and Extended Data Types

The **superstores_demo** database uses *user-defined routines* (UDRs) and extended data types.

A UDR is a routine that you define that can be invoked within an SQL statement or another UDR. A UDR can either return values or not.

The data type system of Dynamic Server is an extensible and flexible system that supports the creation of following kinds of data types:

- Extensions of existing data types, by redefining some of the behavior for data types that the database server provides
- Definitions of customized data types by a user

This section lists the extended data types and UDRs created for the **superstores_demo** database. For information about creating and using UDRs and extended data types, see *Creating User-Defined Routines and User-Defined Data Types*.

The **superstores_demo** database creates the *distinct* data type, PERCENT, in a UDR, as follows:

```
CREATE DISTINCT TYPE percent AS DECIMAL(5,5);
DROP CAST (DECIMAL(5,5) AS percent);
CREATE IMPLICIT CAST (DECIMAL(5,5) AS percent);
```

The **superstores_demo** database creates the following *named row types*:

- **location** hierarchy:
  - **location_t**
  - **loc_us_t**
  - **loc_non_us_t**
- **customer** hierarchy:
  - **name_t**
  - **customer_t**
  - **retail_t**
  - **whlsale_t**
- **orders** table
  - **ship_t**

location_t definition

```
location_id    SERIAL
loc_type       CHAR(2)
company        VARCHAR(20)
street_addr    LIST(VARCHAR(25) NOT NULL)
city           VARCHAR(25)
country        VARCHAR(25)
```

loc_us_t definition

```
state_code     CHAR(2)
zip            ROW(code INTEGER, suffix SMALLINT)
phone          CHAR(18)
```

loc_non_us_t definition

```
province_code  CHAR(2)
zipcode        CHAR(9)
phone          CHAR(15)
```

name_t definition

```
first          VARCHAR(15)
last           VARCHAR(15)
```

customer_t definition

customer_num	SERIAL
customer_type	CHAR(1)
customer_name	name_t
customer_loc	INTEGER
contact_dates	LIST(DATETIME YEAR TO DAY NOT NULL)
cust_discount	percent
credit_status	CHAR(1)

retail_t definition

credit_num	CHAR(19)
expiration	DATE

whlsale_t definition

resale_license	CHAR(15)
terms_net	SMALLINT

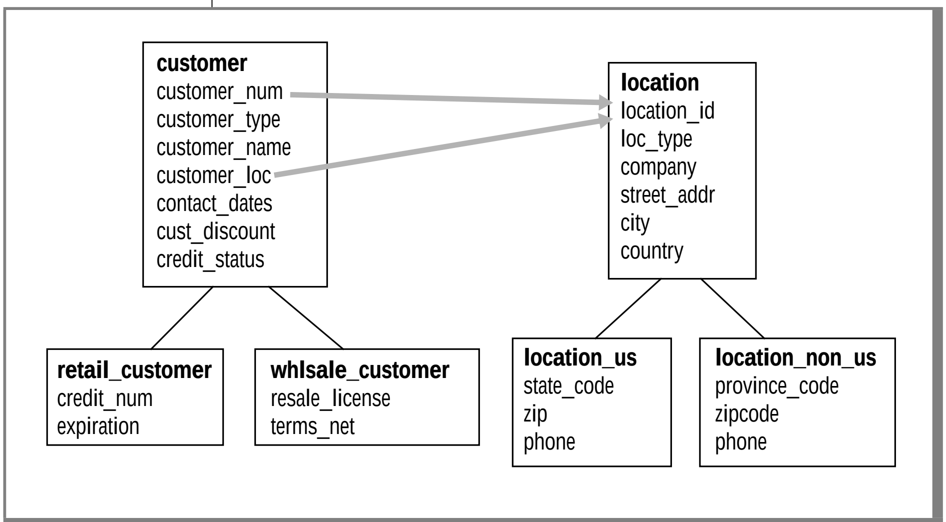
ship_t definition

date	DATE
weight	DECIMAL(8,2)
charge	MONEY(6,2)
instruct	VARCHAR(40)

Table Hierarchies

Figure B-25 shows how the hierarchical tables of the **superstores_demo** database are related.

Figure B-25
Hierarchies of superstores_demo Tables



Referential Relationships

The tables of the **superstores_demo** database are linked by the primary-foreign key relationships that are identified in this section. This type of relationship is called a *referential constraint* because a foreign key in one table *references* the primary key in another table.

The customer and orders Tables

The **customer** table contains a **customer_num** column that holds a number that identifies a customer. The **orders** table also contains a **customer_num** column that stores the number of the customer who placed a particular order. In the **orders** table, the **customer_num** column is a foreign key that references the **customer_num** column in the **customer** table.

The orders and items Tables

The **orders** and **items** tables are linked by an **order_num** column that contains an identification number for each order. If an order includes several items, the same order number appears in several rows of the **items** table. In the **items** table, the **order_num** column is a foreign key that references the **order_num** column in the **orders** table.

The items and stock Tables

The **items** table and the **stock** table are joined by three columns: the **stock_num** column, which stores a stock number for an item, the **manu_code** column, which stores a code that identifies the manufacturer, and the **units** column, which identifies the types of unit in which the item can be ordered. You need the stock number, the manufacturer code, and the units to uniquely identify an item. The same stock number and manufacturer code can appear in more than one row of the **items** table, if the same item belongs to separate orders. In the **items** table, the **stock_num**, **manu_code**, and **unit** columns are foreign keys that reference the **stock_num**, **manu_code**, and **unit** columns in the **stock** table.

The stock and catalog Tables

The **stock** table and **catalog** table are joined by three columns: the **stock_num** column, which stores a stock number for an item, the **manu_code** column, which stores a code that identifies the manufacturer, and the **unit** column, which identifies the type of units in which the item can be ordered. You need all three columns to uniquely identify an item. In the **catalog** table, the **stock_num**, **manu_code**, and **unit** columns are foreign keys that reference the **stock_num**, **manu_code**, and **unit** columns in the **stock** table.

The stock and manufact Tables

The **stock** table and the **manufact** table are joined by the **manu_code** column. The same manufacturer code can appear in more than one row of the **stock** table if the manufacturer produces more than one piece of equipment. In the **stock** table, the **manu_code** column is a foreign key that references the **manu_code** column in the **manufact** table.

The cust_calls and customer Tables

The **cust_calls** table and the **customer** table are joined by the **customer_num** column. The same customer number can appear in more than one row of the **cust_calls** table if the customer calls the distributor more than once with a problem or question. In the **cust_calls** table, the **customer_num** column is a foreign key that references the **customer_num** column in the **customer** table.

The call_type and cust_calls Tables

The **call_type** and **cust_calls** tables are joined by the **call_code** column. The same call code can appear in more than one row of the **cust_calls** table, because many customers can have the same type of problem. In the **cust_calls** table, the **call_code** column is a foreign key that references the **call_code** column in the **call_type** table.

The state and customer Tables

The **state** table and the **customer** table are joined by a column that contains the state code. This column is called **code** in the **state** table and **state** in the **customer** table. If several customers live in the same state, the same state code appears in several rows of the table. In the **customer** table, the **state** column is a foreign key that references the **code** column in the **state** table.

The customer and location Tables

In the **customer** table, the **customer_loc** column is a foreign key that references the **location_id** of the **location** table. The **customer_loc** and **location_id** columns each uniquely identify the customer location.

The manufact and location Tables

The **manu_loc** column in the **manufact** table is a foreign key that references the **location_id** column, which is the primary key in the **location** table. Both **manu_loc** and **location_id** uniquely identify the manufacturer location.

The state and location_us Tables

The **state** and **location_us** tables are joined by the column that contains the state code. The **state_code** column in the **location_us** table is a foreign key that references the **code** column in the **state** table.

The sales_rep and region Tables

The **region_num** column is the primary key in the **region** table. It is a system-generated region number. The **region_num** column in the **sales_rep** table is a foreign key that references and joins the **region_num** column in the region table.

The region and location Tables

The **region_loc** column in the **region** table identifies the regional office location. It is a foreign key that references the **location_id** column in the **location** table, which is a unique identifier for location.

The stock and stock_discount Tables

The **stock** table and the **stock_discount** table are joined by three columns: **stock_num**, **manu_code**, and **unit.** These columns form the primary key for the **stock** table. The **stock_discount** table has no primary key and references the **stock** table.

The stock and units Tables

The **unit_name** column of the **units** table is a primary key that identifies the kinds of units that can be ordered, such as case, pair, box, and so on. The **unit** column of the **stock** table joins the **unit_name** column of the **units** table.

Glossary

Tip: *For additional product-specific information, refer to the glossaries provided in the "Informix Storage Manager Administrator's Guide," the "Informix SNMP Subagent Guide," and other manuals in the Informix documentation set.*

8-bit character A single-byte character that consists of eight bits, which means that the code point is in the range 128 through 255. Examples from the ISO8859-1 code set include the non-English é, ñ, and ö characters. They can be interpreted correctly only if the software that interprets them is 8-bit clean. *See also* non-ASCII character.

8-bit clean An operating system or database server that can process character data that contains 8-bit characters. The operating system or the database server reads the eighth bit as part of the code value. In other words, it does not ignore the eighth bit nor make its own interpretation of the eighth bit.

16-bit code set A code set (such as JIS X0208) in which approximately 65,000 distinct characters can be encoded.

access method A group of *routines* that access or manipulate a table or an index. In the output of a SET EXPLAIN statement, *access method* refers to the mode of table access in a query (for example, SEQUENTIAL SCAN as opposed to INDEX PATH). *See also* primary access method and secondary access method.

access privileges The types of operations that a user has permission to perform in a specific database or database object. Informix maintains its own set of database, table, table-fragment, index, and column-access privileges, which are independent of operating-system access permissions.

active set The collection of rows that satisfies a query associated with a cursor.

aggregate function An SQL function that returns one value for a group of retrieved rows; for example, the frequency, sum, average, maximum, or minimum of an expression in a query or report. *See also* user-defined aggregate.

aggregate support function One of a group of *user-defined functions* that the database server uses to calculate a user-defined aggregate.

alias A temporary alternative name for a table in a query; usually used in complex subqueries and required for a self-join. In a form-specification file or any SQL query, *alias* refers to a single-word alternative name used in place of a qualified table name (for example, **t1** as an alias for *owner.table_name*).

ALS Legacy acronym for Asian Language Support (ALS), *a* feature for multibyte East Asian locales. Supplanted by Global Language Support (GLS).

ANSI Acronym for the American National Standards Institute. This group sets standards in many areas, including the computer industry and standards for the SQL language.

ANSI compliant A database conforming to ANSI/ISO standards for SQL. A database is either ANSI compliant or not ANSI compliant. An ANSI-compliant database enforces certain ANSI requirements, such as implicit transactions, explicit owner naming, no public synonyms, and (for Dynamic Server) unbuffered logging, that are not enforced in databases that are not ANSI compliant.

API *See* application programming interface (API).

application development tool Software, such as INFORMIX-4GL, that you can use to create and maintain a database. The software allows a user to send instructions and data to (and receive information from) the database server.

application process The process that manages an ESQL or other program at runtime. It executes the program logic and initiates SQL requests. Memory that is allocated for program variables, program data, and the fetch buffer is part of this process. *See also* database server process.

application-productivity tools Tools that are used to write applications.

application program An executable file or a logically related set of files.

application programming interface (API) A set of related software components, such as those provided by Informix, that a developer uses to create applications that communicate with a third-party product. An API can include a *library* of *functions*, *header files*, graphic interfaces, and application programs. *See also* SQL API and DataBlade API.

arbitrary rule A series of expressions that you define for expression-based fragmentation, using SQL relational and logical operators. Unlike the range rule, an arbitrary rule allows you to use any relational operator and any logical operator to define the expressions (such as the OR operator to group data).

archiving Copying all the data and indexes of a database onto a new medium, usually a tape or a different physical device from the one that stores the database. Archived material is used for recovering from a failure and is usually performed by a Database Administrator. *See also* backup.

argument A value that is passed to a *routine* or command. *Compare with* parameter.

array An ordered set of items of the same data type. An individual member of the array is referred to as an element and usually is identified by an integer index that gives the position of the element within the array.

ASCII Acronym for American Standards Committee for Information Interchange, a coding scheme that assigns unique integer values to letters, digits, punctuation marks, and certain other nonprintable and printable characters used in computers and telecommunication. It contains 128 characters, each of which can be represented with 7 bits of information. The code set of every Informix locale includes the ASCII characters as a subset. *See also* single-byte character.

ASF Acronym for Associated Services Facility. The code in the ASF portion of Informix products controls the connections between client applications and database servers. System developers use this term; users of Informix products see this term only in occasional error messages.

Asian Language Support (ALS) A class of legacy products that support various multibyte code sets whose characters can require up to 32 bits of storage. ALS servers and tools were available for certain Version 6.x and earlier families of products that were designed to handle East Asian language processing. The functionality of ALS is replaced by Global Language Support (GLS) in current Informix products. For more information about ALS and GLS, see the *Informix Migration Guide*.

attached index An index that is created without an explicit fragmentation strategy. You create an attached index by omitting both the distribution scheme (specified by the FRAGMENT BY clause) and the storage option (specified by the IN clause) of the CREATE INDEX or ALTER FRAGMENT ON INDEX statements. An attached index can be created on a fragmented table.

The location of the index data varies depending on the database server. In most cases, index pages reside in the same tblspaces as the data pages to which they refer.

For Dynamic Server, index pages for user indexes reside in separate tblspaces, but within the same dbspaces, as the data pages to which they refer. Only the **syscatalogs** indexes reside in the same tblspace as the corresponding data pages.

For Extended Parallel Server, both user and system-catalog index pages reside in separate tblspaces but in the same dbspaces as the corresponding data pages. *See also* detached index.

audit event (Not for Extended Parallel Server) Any database server activity or operation that could potentially access and alter data, and which should be recorded and monitored by the database secure auditing facility. Examples of audit events include accessing tables, altering indexes, dropping chunks, granting database access, updating the current row, running database utilities, and so forth. (For a complete list of audit events, see the *Trusted Facility Manual*.)

audit file (Not for Extended Parallel Server) A file that contains records of audit events and resides in the specified audit directory. Audit files form an audit trail of information that can be extracted by the database secure auditing facility for analysis by the database administrator.

audit mask (Not for Extended Parallel Server) A structure that specifies which events should be audited (or excluded from auditing) by the database secure auditing facility.

auxiliary statements The SQL statements that you use to obtain auxiliary information about tables and databases. These statements include INFO, OUTPUT, WHENEVER, and GET DIAGNOSTICS.

B-tree A method of organizing an index into a tree structure for efficient record retrieval.

B-tree index A type of index that uses a balanced tree structure for efficient record retrieval. A B-tree index is balanced when the leaf nodes are all at the same level from the root node. B-tree indexes store a list of rowids for any duplicate key value data in ascending or descending order. *See also* bitmap index and R-tree index.

backup A duplicate of a computer file on another device or tape to preserve existing work, in case of a computer failure or other mishap. A *backup* refers to duplicating logical-log files while archiving refers to duplicating data.

base table *See* table.

base type *See* opaque data type.

before-image The image of a row, page, or other item before any changes are made to it.

big-endian A hardware-determined storage method in which the most-significant byte of a multibyte number has the lowest address. *See also* little-endian.

bitmap index A type of index that stores a bitmap for any highly duplicate key value. The bitmap indicates which rows have the duplicate key value. You create a bitmap index with the USING BITMAP keywords in the CREATE INDEX statement. *See also* B-tree index.

blob A legacy term for a large object that is now known as a simple large object and includes TEXT or BYTE data types. These data objects effectively have no maximum size (theoretically as large as 2^{31} bytes). *See also* simple large object.

BLOB Acronym for binary large object. A data type for a *smart large object* that stores any type of binary data, including images. It can be stored and retrieved in pieces and has database properties such as recovery and transaction rollback. *See also* CLOB.

blobpage (Not for Extended Parallel Server) The unit of disk allocation within a blobspace. The database server administrator determines the size of a blobpage. The size can vary, depending on the size of the TEXT or BYTE data that the user inserts.

blobspace (Not for Extended Parallel Server) A logical collection of *chunks* that is used to store TEXT and BYTE data.

Boolean A variable or an expression that can take on the logical values TRUE (1), FALSE (0), or UNKNOWN (if NULL values are involved).

BOOLEAN A built-in data type that supports single-byte true / false values. TRUE is represented internally as `0` and externally as `t`. FALSE is represented internally as `1` and externally as `f`. A NULL value is represented as NULL.

Boolean function A function that returns a Boolean value (true or false). A Boolean function can act as a *filter*.

branch node An index page that contains pointers to a leaf node or other branch nodes. The database server creates a branch node when the root node and subsequent leaf nodes become full.

buffer A portion of computer memory where a program temporarily stores data. Data typically is read into or written out from buffers to disk.

buffered disk I/O Disk I/O that the operating system controls instead of an application. With buffered disk I/O, the operating system stores data in the kernel portion of memory before periodically writing the data to disk. *See also* unbuffered disk I/O and disk I/O.

buffered logging A type of logging that holds transactions in a memory buffer until the buffer is full, regardless of when the transaction is committed or rolled back. Informix database servers provide this option to speed up operations by reducing the number of disk writes.

built-in Provided by the database server, usually in the system catalog; not defined by the user.

built-in data type A predefined data type that the database server supports; for example, INTEGER, CHAR, or SERIAL.

byte An physical computer storage unit of 8 bits. A character is not necessarily one byte. In a multibyte code set, a character can require more than one byte.

BYTE A data type for a simple large object *that stores any type of binary data and can be as large as* 2^{31} *bytes. See also* TEXT.

Cartesian product The set that results when you pair each and every member of one set with each and every member of another set. A Cartesian product results from a multiple-table query when you do not specify the joining conditions among tables. *See also* join.

cascading deletes Deletion of rows from a child table that were associated by foreign key to a row that is deleted from the parent table. When any rows are deleted from the primary key column of a table, cascading deletes, if enabled, eliminate identical information from any foreign-key column in a related table.

case sensitivity The condition of distinguishing between uppercase and lowercase letters. Be careful running Informix programs, because certain commands and their options are case sensitive; that is, they react differently to the same letters presented in uppercase and lowercase characters. In non-default locales, the locale files specify which characters (if any) are uppercase or lowercase.

cast A mechanism that converts one data type to another. *See also* built-in cast, user-defined cast, explicit cast, and implicit cast.

cast function A user-defined function that implements a cast. The function must be registered with the CREATE CAST statement before it can be used.

character A logical unit of storage for a code point. A character is equal to one or more bytes and can be numeric, alphabetic, or a nonprintable character (such as a control character). *See also* multibyte character and single-byte character.

character set One or more natural-language alphabets, together with additional symbols for digits, punctuation, ligatures, diacritical marks, and whitespace. (A *natural language* is a written language that human beings use to communicate with each other, such as English, Chinese, or German.) *See also* code set.

character special device *See* unbuffered disk I/O.

check constraint A logical condition that must be satisfied before data values can be entered into a column of a database table during an INSERT or UPDATE statement.

checkpoint A point in time during a database server operation when the pages on disk are synchronized with the pages in the shared memory buffer pool. It can be a *full checkpoint* or a *fuzzy checkpoint*.

child table The referencing table in a referential constraint. *See also* parent table.

chunk The largest contiguous section of disk space available for a database server. A specified set of chunks defines a *dbspace* or *blobspace*. A database server administrator allocates a chunk to a dbspace or blobspace when that dbspace or blobspace approaches full capacity. A chunk contains a certain number of pages. (Extended Parallel Server does not support blobspace chunks.)

client application A program that requests services from a server program, typically a file server or a database server. For the GLS feature, the term *client application* includes database server utilities.

client computer The computer on which a client application runs.

client locale The locale that a client application uses to perform read and write operations on the client computer. The **CLIENT_LOCALE** environment variable can specify a nondefault locale. *See also* server locale.

client/server architecture A hardware and software design that allows the user interface and database server to reside on separate nodes or platforms on a single computer or over a network. See also ASF, client application, and server-processing locale.

client/server connection statements The SQL statements that can make connections with databases. These statements include CONNECT, DISCONNECT, and SET CONNECTION.

CLOB Acronym for character large object. A data type for a smart large object that stores large text items, such as PostScript or HTML files. It can be stored and retrieved in pieces and has database properties such as recovery and transaction rollback. *See also* BLOB.

close a cursor To drop the association between a cursor and the active set of rows that results from a query.

close a database To deactivate the connection between a client application and a database. Only one database can be active at a time.

close a file To deactivate the association between a file and a program.

cluster an index To rearrange the physical data of a table according to a specific index.

cluster key The column in a table that logically relates a group of simple large objects or smart large objects that are stored in an optical cluster.

clustersize The amount of space, specified in kilobytes, that is allocated to an optical cluster on an optical volume.

code point A bit pattern that represents one character in a code set. For example, in the ASCII code set, the uppercase *A* character has a code point of `0x41`.

code set The representation of a character set that specifies how to map each element of a character set to a unique code point. For example, ASCII, ISO8859-1, Microsoft 1252, and EBCDIC are code sets to represent the English language. A locale name specifies a code set.

code-set conversion The process of converting character data from one code set (the *source* code set) to another (the *target* code set). Code-set conversion is useful when the client and server computers use different code sets to represent the same character data.

code-set order The serialized order of characters within a code set. For example, in the ASCII code set, uppercase characters (*A* through *Z*) are ordered before lowercase characters (*a* through *z*). *See also* collation order and localized order.

cogroup A named group of coservers. At initialization, the database server creates a cogroup that is named **cogroup_all** from all configured *coservers*.

collating sequence The logical order in which the character-string values in a database are sorted and indexed.This is based on either the order of the code set or else some locale-specific order. Collating sequence is also known as *collation order.*

collation The process of sorting character strings according to some order.

collation order *See* collating sequence and code-set order.

collection An instance of a *collection data type*; a group of *elements* of the same *data type* stored in a SET, MULTISET, or LIST.

collection cursor A database cursor that has an Informix ESQL/C *collection variable* associated with it and provides access to the individual *elements* of a column whose data type is a *collection data type.*

collection data type A complex data type whose instances are groups of *elements* of the same *data type,* which can be any *opaque data type*, *distinct data type*, *built-in data type*, *collection data type*, or *row type*.

collection-derived table A table that Informix ESQL/C or SPL creates for a collection column when it encounters the TABLE keyword in an INSERT, DELETE, UPDATE, or SELECT statement. ESQL/C and SPL store this table in a collection variable to access *elements* of the collection as rows of the collection-derived table.

collection subquery A query that takes the result of a subquery and turns it into a expression by using the MULTISET keyword to convert returned values into a MULTISET collection.

collection variable An Informix ESQL/C *host variable* or *SPL variable* that holds an entire *collection* and provides access, through a *collection cursor,* to the individual *elements* of the collection.

collocated join A join that occurs locally on the coserver where the data resides. The local coserver sends the data to the other coservers after the join is complete.

column A data element that contains a specified type of information that occurs in every row of the table. Also known as a *display label* or a *field*. *See also* row.

column expression An expression that includes a column name and optionally uses *column subscripts* to define a *column substring*.

column subscript A subscript in a column expression.

column substring A substring in a column expression.

command file A system file that contains one or more statements or commands, such as SQL statements.

comment Information in a program file that is not processed by the computer but that documents the program. Special characters such as the sharp sign (#), braces ({ }), slash marks (/) asterisks (*), or a double hyphen (--) can be used to identify comments, depending on the programming context.

commit work To complete a transaction by accepting all changes to the database since the beginning of the transaction. *See also* roll back.

Committed Read An Informix level of isolation in which the user can view only rows that are currently committed at the moment when the query is requested; the user cannot view rows that were changed as a part of a currently uncommitted transaction. Committed Read is available through a database server and set with the SET ISOLATION statement. It is the default level of isolation for databases that are not ANSI compliant. *See also* Read Committed.

compile To translate source code (in a high-level language) into executable code. *Compare with* execute and link. *See also* source file.

compile-time error An error that occurs when you *compile* the program source code. This type of error indicates syntax errors in the source code. *Compare with* runtime error.

complex data type A *data type* that is built from a combination of other data types using an SQL type constructor and whose components can be accessed through SQL statements. *See also* row type and collection data type.

component In the High-Performance Loader (HPL), the information required to load or unload data is organized in several *components*. The components are format, map, filter, query, project, device array, load job, and unload job.

composite data type *See* row type.

composite index An index constructed on two or more columns of a table. The ordering imposed by the composite index varies least frequently on the first-named column and most frequently on the last-named column.

composite join A join between two or more tables based on the relationship among two or more columns in each table. *See also* simple join.

compressed bitmap An indexing method that identifies records through a fragment identifier and a record identifier.

concatenate To append a second string to the end of a first string.

concatenation operator An operator whose notation is composed of two pipe symbols (||); this is used in expressions to indicate the joining of two strings.

concurrency Access by two or more processes to the same database simultaneously.

configuration management (CM) coserver A coserver that Informix designates to run CM software and store CM data.

configuration file A file read during database server disk or shared-memory initialization that contains the parameters that specify values for configurable behavior. Database server and its archiving tool use configuration files.

connection A logical association between two applications or between an application and a database environment, created by a CONNECT or DATABASE statement. Database servers can also have connections to one another. *See also* explicit connection, implicit connection, and multiplexed connection.

connection coserver The coserver to which a client is directly connected. *See also* coserver and participating coserver.

connection redirector An Extended Parallel Server feature, enabled by a setting in the **sqlhosts** file, whereby the database server attempts to establish a client connection with each coserver in a dbserver group until a connection succeeds.

constant A value that cannot change during the execution of a program or command. In some programming languages, a constant has a name that can be referenced. *Compare with* variable. *See also* literal.

constraint A restriction on what kinds of data can be inserted or updated in tables. *See also* check constraint, primary-key constraint, referential constraint, not-null constraint, and unique constraint.

constructed data type *See* complex data type.

constructor *See* type constructor.

control character A character whose occurrence in a particular context initiates, modifies, or stops a control function (an operation to control a device, for example, in moving a visual cursor or in reading data). In a program, you can define actions that use the CTRL key with another key to execute some programming action (for example, entering CTRL-W to obtain online Help in Informix products). A control character is sometimes referred to as a *control key.* *Compare with* printable character.

cooked files *See* buffered disk I/O.

coordinating server In a query that spans multiple database servers, the server in which the query is initiated is called the *coordinator* or *coordinating server.* This server is also sometimes called the *local server* because it is the local server to the client initiating the query. To respond to the query, the coordinating server starts sessions on the other servers involved in the query. *See also* distributed query, subordinate servers, and remote servers.

correlated subquery A subquery (or inner SELECT) that depends on a value produced by the outer SELECT statement that contains it. Also a nested subquery whose WHERE clause refers to an attribute of a relation that is declared in an outer SELECT. Correlated subqueries reference one or more columns from one or more tables of a parent query and need to be evaluated once for each row in the parent query. *See also* independent subquery and subquery.

correlation name The prefix used with a column name in a triggered action to refer to an old (before triggering statement) or a new (after triggering statement) column value. The associated column must be in the triggering table. *See also* trigger.

corrupted database A database whose tables or indexes contain incomplete, inconsistent, or invalid data.

corrupted index An index that does not correspond exactly to the data in its table.

coserver The functional equivalent of a database server that operates on a single node. *See also* connection coserver and participating coserver.

current row The most recently retrieved row of the active set of a query.

cross-server query *See* distributed query.

cursor In SQL, an identifier associated with a group of rows or with a collection data type. Conceptually, the pointer to the current row or collection element. You can use cursors for SELECT statements or EXECUTE PROCEDURE statements (associating the cursor with the rows returned by a query) or INSERT statements (associating the cursor with a buffer to insert multiple rows as a group). A select cursor is declared for sequential only (regular cursor) or nonsequential (scroll cursor) retrieval of rows. In addition, you can declare a select cursor for update (initiating locking control for updated and deleted rows) or WITH HOLD (so that completing the current transaction does not close the cursor). In ESQL/C, a cursor can be dynamic, meaning that it can be referenced by an identifier or by a character variable.

cursor function A *user-defined routine* (UDR) that returns one or more rows of data and therefore requires a cursor to execute. An *SPL routine* is a cursor function when its RETURN statement contains the WITH RESUME keywords. An *external function* is a cursor function when it is defined as an iterator function. *Compare with* noncursor function.

cursor manipulation statements The SQL statements that control cursors; specifically, the CLOSE, DECLARE, FETCH, FLUSH, OPEN, and PUT statements.

Cursor Stability An Informix level of isolation available through the SET ISOLATION statement in which the database server must secure a shared lock on a fetched row before the row can be viewed. The database server retains the lock until it receives a request to fetch a new row. *See also* isolation.

data access statements The subset of SQL statements that you can use to grant and revoke permissions and to lock tables.

data definition statements The subset of SQL statements (sometimes called *data definition language*, or DDL) to create, alter, drop, and rename data objects, including databases, tables, views, synonyms, triggers, sequences, and user-defined routines.

data dictionary The set of tables that keeps track of the structure of the database and the inventory of database objects. This is also called the system catalog. Each database that a database server supports has its own system catalog.

data distribution A mapping of the data values within a column into a set of equivalence categories, equivalent to a histogram or to a frequency distribution. You can use the UPDATE STATISTICS statement (specifying the MEDIUM or HIGH keyword options) to create data distributions.

data integrity The process of ensuring that data corruption does not occur when multiple users simultaneously try to alter the same data. Locking, constraints, and transaction logging are used to control data integrity.

data integrity statements SQL statements that you use to control transactions and audits. Data integrity statements also include statements for repairing and recovering tables.

data manipulation statements The SQL statements that can query tables, insert into tables, delete from tables, or update tables (SELECT, INSERT, DELETE, UPDATE). The LOAD and UNLOAD utilities also are sometimes called data manipulation statements.

data partitioning *See* table fragmentation.

data replication The ability to allow database objects to have more than one representation at more than one distinct site.

data restriction Synonym for *constraint*.

data type A descriptor assigned to each column in a table or program variable, which indicates the type of data the column or program variable is intended to hold. Informix data types are discussed in Chapter 2, "Data Types." Informix data types for Global Language Support are discussed in the *Informix Guide to GLS Functionality*. *See also* built-in data type, complex data type, distinct data type, opaque data type, and user-defined data type.

database A collection of information (contained in tables) that is useful to some organization or that is used for a specific purpose. See also relational database.

Database Administrator *See* DBA.

database application A program that applies database management techniques to implement specific data manipulation and reporting tasks.

database environment Used in the CONNECT statement, either the database server or the database server-and-database to which a user connects.

database locale The locale that a database server uses for the code set, collation, and date, time, and currency display conventions of a database. The **DB_LOCALE** environment variable can specify a nondefault database locale. *See also* locale.

database management system *See* DBMS.

database object An SQL entity that is recorded in a system catalog table, such as a *table*, column, *constraint*, access method, *default value*, *dependency*, *index*, *operator class*, *prepared statement*, *privilege*, *role*, *sequence*, *synonym*, *trigger*, *user-defined aggregate*, *user-defined cast*, *user-defined data type*, *user-defined routine*, or *view*.

database server A software package that manages access to one or more databases for one or more client applications. *See also* relational database server.

database server process A virtual processor that functions similarly to a CPU in a computer. *See also* application process.

database server utility A program that performs a specific task. For example, DB-Access, **dbexport**, and **onmode** are Informix database server utilities.

DataBlade API An application programming interface (API) that Informix provides to allow a C *user-defined routine* access to the client application. You can also use the DataBlade API to create client LIBMI applications (for backward compatibility with Illustra applications).

DataBlade module A group of *database objects* and supporting code that extends an object-relational database to manage new kinds of data or add new features. A DataBlade module can include new data types, *routines*, *casts*, *aggregates*, *access methods*, SQL code, client code, and installation programs.

DBA Acronym for *Database Administrator*. The DBA is responsible for the contents and use of a database, whereas the database server administrator (DBSA) is responsible for managing one or more database servers. Also a level of privilege, typically for operations that most users are not authorized to perform.

DBA-PRIVILEGED A class of SPL routines that only a user with DBA database privileges creates.

DBMS Acronym for *database management system*. These are all the components necessary to create and maintain a database, including the application development tools and the database server.

dbserver group A collection of coservers defined and named by entries in the **sqlhosts** file. Dbserver groups make multiple coservers into a single logical entity for establishing or changing client/server connections.

dbslice A named set of dbspaces that can span multiple coservers. A dbslice is managed as a single storage object. *See also* logslice, physslice, and rootslice.

dbspace A logical collection of one or more chunks. Because chunks represent specific regions of disk space, the creators of databases and tables can control where their data is physically located by placing databases or tables in specific dbspaces. A dbspace provides a link between logical (such as tables) and physical (such as chunks) units of storage. *See also* root dbspace.

DDL Acronym for data definition language, a subset of the Structured Query Language (SQL) for declaring, modifying, and dropping database objects (such as tables, constraints, or indexes). *See also* data definition statements.

deadlock A situation in which two or more threads cannot proceed because each is waiting for data locked by the other (or another) thread. The database server monitors and prevents potential deadlock situations by sending an error message to the application if a request for a lock might result in a deadlock.

debug file A file that receives output used for debugging purposes.

decision-support application An application that provides information that is used for strategic planning, decision-making, and reporting. It typically executes in a batch environment in a sequential scan fashion and returns a large fraction of the rows scanned. Decision-support queries typically scan the entire database. *See also* online transaction processing application.

decision-support query A query that a decision-support application generates. It often requires multiple joins, temporary tables, and extensive calculations, and can benefit significantly from PDQ. *See also* online transaction processing queries.

declaration statement A programming language statement that describes or defines objects; for example, defining a program variable. *Compare with* procedure. *See also* data definition statements.

default Values or behavior that take effect unless the user explicitly specifies another value or action.

default locale The locale that a product uses unless you specify a different (nondefault) locale. For Informix products, U.S. English is the default locale.

default value A value that is used when no explicit value is specified. For example, you can assign default values to columns with the ALTER TABLE and CREATE TABLE statements and to variables in *SPL routines*.

delete To remove any row or combination of rows with the DELETE statement.

delimited identifier If the **DELIMIDENT** environment variable is set, this is an SQL identifier enclosed between double (") quotation marks. This supports identifiers that are SQL-reserved keywords or that contain whitespace or other characters outside the default SQL character set for identifiers.

delimiter A character used as a boundary on a field or as the terminator for a column or row. Some files and prepared objects require semicolon (;), comma (,), blank space ( ), or tab delimiters between statements. Statements of SQL can have semicolon or other delimiters in some programming contexts.

deluxe mode A method of loading or unloading data that uses regular inserts.

descriptor A quoted string or variable that identifies an allocated system-descriptor area or an sqlda structure. It is used for the Informix SQL APIs. *See also* identifier.

detached index As index whose distribution scheme (specified by the FRAGMENT BY clause) and storage option (specified by the IN clause) of the CREATE INDEX or ALTER FRAGMENT ON INDEX statement differ from the distribution scheme of the underlying table. Index pages reside in separate dbspaces from the corresponding data pages. *Compare with* attached index.

device array A list of I/O devices. *See also* component.

diagnostic area A data structure (sometimes called sqlda) that stores diagnostic information about an executed SQL statement.

diagnostics table A special table that holds information about the integrity violations caused by each row in a violations table. You use the START VIOLATIONS TABLE statement to create violations tables and diagnostics tables and associate them with a base table.

Dirty Read An Informix isolation level set with the SET ISOLATION statement that disregards locks and allows viewing of any existing rows, even rows that currently can be altered from inside an uncommitted transaction. Dirty Read is the lowest level of isolation (no isolation at all), and is thus the most efficient. *See also* Read Uncommitted.

disabled mode The object mode in which a database object is disabled. When a constraint, index, or trigger is in the disabled mode, the database server acts as if the object does not exist and does not take it into consideration during the execution of data manipulation statements.

disk configuration The organization of data on a disk; also refers to the process of preparing a disk to store data.

disk I/O The process of transferring data between memory and disk. The I/O refers to input/output.

display label A temporary name for a column or expression in a query.

distinct data type A *data type* that you declare with the CREATE DISTINCT TYPE statement. A distinct data type has the same internal storage representation as its *source type* (an existing *opaque data type, built-in data type, named row type,* or distinct data type*)* but a different name, and can have different casts and routines. To compare a distinct data type with its source type requires an *explicit cast.* A distinct data type inherits all routines that are defined on its source type.

distributed query A query that accesses data from a database other than the current database.

distribution *See* data distribution.

distribution scheme *See* table fragmentation.

DLL See dynamic link library (DLL).

DML Acronym for data manipulation language. *See also* data manipulation statements.

dominant table *See* outer join.

DRDA Acronym for Distributed Relational Database Architecture. DRDA is an IBM-defined set of protocols that software manufacturers can follow to develop connectivity solutions between heterogeneous relational database management environments.

DSS Acronym for Decision Support System. *See also* decision-support application.

duplicate index An index that allows duplicate values in the indexed column.

dynamic link library (DLL) A *shared-object file* on a Windows system. *See also* shared library.

dynamic management statements The SQL statements that describe, execute, and prepare other statements.

dynamic routine-name specification The execution of a *user-defined routine* whose name is determined at runtime through an *SPL variable* in the EXECUTE PROCEDURE, EXECUTE ROUTINE, or EXECUTE FUNCTION statement.

Dynamic Server instance The set of processes, storage spaces, and shared memory that together comprise a complete database server. A single Dynamic Server instance can support more than one database.

dynamic SQL The statements and structures that allow a program to form an SQL statement at runtime, so that portions of the statement can be determined by user input.

dynamic statements The SQL statements that are specified at runtime (when the program is executed), rather than when the program is compiled. You can use the PREPARE statement to create dynamic SQL statements.

EBCDIC Acronym for *Extended Binary Coded Decimal Interchange Code*, a 256-element 8-bit character set that was originally designed for mainframe computers.

element A member of a *collection*, such as a LIST, MULTISET, or SET data type. An element can be a value of any *built-in data type*, *opaque data type*, *distinct data type*, *named row type*, *unnamed row type*, or *collection data type*.

element type The *data type* of the *elements* in a *collection*.

embedded SQL The SQL statements that are placed within some other host language. For example, Informix supports embedded SQL in C.

enabled mode The default object mode of database objects. When a constraint, index, or trigger is in this mode, the database server recognizes the existence of the object and takes the object into consideration while executing data manipulation statements. *See also* object mode.

end-user format The format in which data appears within a client application as literal strings or character variables. End-user formats are useful for data types whose database format is different from the format to which users are accustomed.

end-user routine A *user-defined routine* (UDR) that performs a task within an SQL statement that the existing *built-in* routines do not perform. Examples of tasks include encapsulating multiple SQL statements, creating trigger actions, and restricting who can access *database objects*.

environment variable A variable that the operating system maintains for each user and makes available to all programs that the user runs.

error log A file that receives error information whenever a program runs.

error message A message that is associated with a (usually negative) number. Informix applications display error messages on the screen or write them to files.

error trapping *See* exception handling.

escape character A character that indicates that the next character, normally interpreted by the program as having special significance, is to be processed as a literal character instead. The escape character precedes the special character (such as a wildcard or delimiter) to "escape" (that is, ignore) the special significance.

escape key The physical key of a keyboard, usually marked ESC, that is used to terminate one mode and start another mode in most UNIX and DOS systems.

ESQL/C Smart Large-Object API An *API* of C routines that an Informix ESQL/C client application can use to access *smart large objects* as operating-system files. The ESQL/C Smart Large-Object API is part of the Informix ESQL/C *SQL API*. You can also access smart large objects with a set of functions in the *DataBlade API*.

exception An error or warning that the database server returns, or a state that a SPL statement initiates.

exception handling The code in a program that anticipates and reacts to runtime errors and warnings. Also referred to as error handling or error trapping.

exclusive access Sole access to a database or table by a user. Other users are prevented from using it.

exclusive lock A lock on an object (row, page, table, or database) that is held by a single thread that prevents other processes from acquiring a lock of any kind on the same object.

executable file A file that contains code that can be executed directly. A C-language object file can be an executable file; it contains the machine-level instructions that correspond to the C-language *source file*.

execute To run a statement, program, *routine*, or a set of instructions. *See also* executable file.

explicit cast A *user-defined cast* that a user explicitly invokes with the CAST AS keyword or cast operator (**::**). *See also* implicit cast.

explicit connection A connection made to a database environment that uses the CONNECT statement. *See also* implicit connection.

explicit select list A SELECT statement in which the user explicitly specifies the names of the columns from which the query returns values.

explicit transaction A transaction that is initiated by the BEGIN WORK statement. This type of transaction is available only in non-ANSI compliant databases that support logging. *See also* implicit transaction and singleton implicit transaction.

exponent The power to which a value is to be raised.

express mode An Extended Parallel Server method of loading or unloading data that uses light appends.

expression Anything from a simple numeric or alphabetic constant to a more complex series of column values, functions, quoted strings, operators, and keywords. A Boolean expression contains a logical operator (>, <, =, !=, IS NULL, and so on) and evaluates as TRUE, FALSE, or UNKNOWN. An arithmetic expression contains the operators (+, –, ×, / , and so on) and evaluates as a number.

expression-based fragmentation A distribution scheme that distributes rows to fragments according to a user-specified expression that is defined in the WHERE clause of an SQL statement.

extended data type A term used to refer to data types that are not built in; namely *complex data type*s, *opaque data type*s, and *distinct data typ*es.

extent A continuous segment of disk space that a database server allocated to a tblspace (a table). The user can specify both the initial extent size for a table and the size of all subsequent extents that a database server allocates to the table.

external function An *external routine* that returns a single value.

external procedure An *external routine* that does not return a value.

external routine A *user-defined routine* that is written in an external language that the database supports. These external languages include the C and Java languages. The routine names, parameters, and other information are registered in the system catalog tables of a database. However, the executable code of an external routine is stored outside the database. An external routine can be an *external function* or an *external procedure*.

external space Storage space that a user-defined access method manages rather than the database server. The IN clause of the CREATE TABLE and CREATE INDEX statements can specify the name of an external space instead of a dbspace.

external table A database table that is not in the current database. It might or might not be in a database that the same database server manages.

extspace (Not for Extended Parallel Server) A logical name associated with an arbitrary string that signifies the location of external data. Access its contents with a user-defined access method.

family name A quoted string constant that specifies a family name in the optical family. *See also* optical family.

fault tolerance *See* high availability.

fetch The action of moving a cursor to a new row and retrieving the row values into memory.

fetch buffer A buffer in the application process that the database server uses to send fetched row data (except TEXT and BYTE data) to the application.

field A component of a *named row type* or *unnamed row type* that contains a name and a *data type* and can be accessed in an SQL statement by using dot notation in the form *row type name.field name*. *See also* column.

file A collection of related information stored together on a system, such as the words in a letter or report, a computer program, or a listing of data.

file server A network node that manages a set of disks and provides storage services to computers on the network.

filename extension The part of a filename following the period. For example, DB-Access appends the extension **.sql** to command files.

filter A set of conditions (sometimes called a *predicate*) for selecting rows or records. In an SQL query, the conditional expression in the WHERE clause is a filter that controls the active set of the query. The High-Performance Loader (HPL) uses a filter component to screen data before loading it into a database.

filtering mode An object mode of a database object, causing bad rows to be written to the violations table during DML operations. During DML operations, the database server enforces requirements of a constraint or of a unique index that is in this mode, and identifies any rows that would violate the requirement.

fixchar A character data type in ESQL/C programs, for fixed-length character strings that are padded (as needed) with trailing blanks, and not null-terminated.

fixed-point number A number where the decimal point is fixed at a specific place regardless of the value of the number.

flag A command-line option, usually indicated by a minus (-) sign in UNIX systems. For example, in DB-Access the **-e** flag echoes input to the screen.

flexible temporary table An explicit temporary table that Extended Parallel Server automatically fragments using a round-robin distribution scheme.

floating-point number A number with fixed precision (total number of digits) and undefined scale (number of digits to the left of the decimal point). The decimal point *floats* as appropriate to represent an assigned value.

foreign key A column or set of columns that references a unique or primary key in a table. For every entry in a foreign-key column containing only non-NULL values, there must exist a matching entry in the unique or primary column.

format A description of the organization of a data file. *See also* component.

formatting character For XPS, a percent sign (%) followed by a letter (c, n, o, or r). In a command line, Extended Parallel Server expands the formatting character to designate multiple coserver numbers (%c), multiple nodes (%n), multiple ordinal numbers designating dbspaces (%d), or a range of dbspaces (%r).

fragment *See* index fragment and table fragment.

fragment elimination The process of applying a filter predicate to the fragmentation strategy of a table or index and removing the fragments that do not apply to the operation.

fragmentation The process of defining groups of rows within a table based on a rule and then storing these groups, or fragments, in separate dbspaces that you specify when you create a table or index fragmentation strategy.

full checkpoint A type of checkpoint where the pages on disk are synchronized with the pages in the shared-memory buffer pool.

function A *routine* that returns one or more values. *See also* user-defined function.

function cursor A cursor that is associated with an EXECUTE FUNCTION statement, which executes routines that return values. *See also* cursor function.

function overloading See routine overloading.

fuzzy checkpoint A type of checkpoint where only certain pages on disk are synchronized with the pages in the shared-memory buffer pool, and the logical log is used to synchronize the rest of the pages during fast recovery.

gateway A device that establishes data communications between networks.

generalized-key (GK) index A type of index for static tables with Extended Parallel Server that can speed certain queries by storing the result of an expression as a key in a B+ tree or bitmap index. Three types of GK index are *selective, virtual column*, and *join*.

gigabyte A unit of storage, equal to 1024 megabytes or 1024^3 bytes.

Global Language Support (GLS) A feature that enables Informix APIs and database servers to support different languages, cultural conventions, and code sets. For information about the GLS feature, see the *Informix Guide to GLS Functionality*.

global variable A *variable* or *identifier* whose scope of reference is all modules of a program. *Compare with* local variable.

globally detached index For Extended Parallel Server, a type of index that has a fragmentation strategy that is independent of the table fragmentation and where the database server cannot verify that each index row resides on the same coserver as the referenced data row. You can use an expression, system-defined hash, or hybrid distribution scheme to create globally detached indexes for any table. *See also* locally detached index.

GLS *See* Global Language Support (GLS).

GLS API A legacy acronym for Informix GLS. An *API* of C routines that a C-language *external routine* can use to access Informix GLS *locales*. This API also includes functions that obtain culture-specific collation, and time, date, number, and currency, formats, and functions that provide a uniform way of accessing character data, regardless of whether the locale supports *single-byte characters* or *multibyte characters*.

hash fragmentation *See* system-defined hash fragmentation.

hash rule A user-defined algorithm that maps each row in a table to a set of hash values and that is used to determine the fragment in which a row is stored.

header file A *source file* that contains *declarations* for *variables*, *constants*, and *macros* that a particular group of modules or programs share.

help message Online text displayed automatically or at the request of the user to assist the user in interactive programs. Such messages are stored in help files.

heterogeneous commit A protocol governing a group of database servers, of which at least one is a *gateway* participant. It ensures the all-or-nothing basis of distributed transactions in a heterogeneous environment. *See also* two-phase commit.

hierarchy A tree-like data structure in which some groups of data are subordinate to others such that only one group (called **root**) exists at the highest level, and each group except root is related to only one parent group on a higher level.

high availability The ability of a system to resist failure and data loss. High availability includes features such as fast recovery and mirroring. It is sometimes referred to as *fault tolerance*.

High-Performance Loader The High-Performance Loader (HPL) utility is part of Dynamic Server. The HPL loads and unloads data using parallel access to devices. *See also* external table.

highlight A rectangular inverse-video area marking your place on the screen. A highlight can indicate the current option on a menu, or the current character in an editing session. If a terminal cannot display highlighting, the current option can appear in angle (**< >**) brackets, with the current character underlined.

hold cursor A cursor that is created using the WITH HOLD keywords. A hold cursor remains open past the end of a transaction. It allows uninterrupted access to a set of rows across multiple transactions.

home page The page that contains the first byte of the data row, specified by the rowid. Even if a data row outgrows its original storage location, the home page does not change. The home page contains a forward pointer to the new location of the data row. *See also* remainder page.

host variable An SQL API program variable that you use in an embedded statement to transfer information between the SQL API program and the database.

HPL *See* High-Performance Loader.

hybrid fragmentation A distribution scheme that lets the user specify two fragmentation methods. Usually one method is used globally and one method is used locally.

identifier In the default locale, a sequence of letters, digits, and underscores (_) that is the unqualified name of a database, storage, or program object. (Additional characters are valid in other locales, or if the **DELIMIDENT** variable is set.)

implicit transaction A transaction that begins implicitly after the COMMIT WORK or ROLLBACK WORK statement. This is the only type of transaction that ANSI compliant databases support, but is also available for other databases that support logging. *See also* explicit transaction and singleton implicit transaction.

implicit cast A *built-in* or *user-defined cast* that the database server automatically invokes to perform data-type conversion. *See also* explicit cast.

implicit connection A connection that is made using the DATABASE, CREATE DATABASE, START DATABASE, or DROP DATABASE statement. *See also* explicit connection.

implicit select list A SELECT statement that uses the asterisk (*) wildcard symbol so that a query returns values from all columns of the table.

incremental archiving A system of archiving that allows the option to archive only those parts of the data that have changed since the last archive was created.

independent subquery A subquery that has no relationship to or dependency on any of its parent queries. It needs to be evaluated only once and the results can be used thereafter. In independent subqueries, both the parent and subquery are parallelized. *See also* correlated subquery and subquery.

index A structure of entries, each of which contains a value or values and a pointer to the corresponding location in a table or smart large object. An index might improve the performance of database queries by ordering a table according to key column values or by providing access to data inside of large objects.

index fragment Consists of zero or more index items grouped together, which can be stored in the same dbspace as the associated table fragment or in a separate dbspace. An index fragment also might occupy an sbspace or an *extspace*.

Informix user ID A login user ID (login *user name* or *authorization identifier*) that must be valid on all computer systems (operating systems) involved in the client's database access. Often referred to as the *client's user ID*. This need not refer to a fully functional user account on the computer system; only the user identity components of the user account information are significant to Informix database servers. A given user typically has the same Informix user ID on all networked computer systems involved in the database access. The authorization identifier **informix** is required for some database objects and operations.

Informix user password A user ID password that must be valid on all computer systems (operating systems) involved in the client's database access. When the client specifies an explicit user ID, most computer systems require the Informix user password.

inheritance The process that allows an object to acquire the properties of another object. Inheritance allows for incremental modification, so that an object can inherit a general set of properties and add properties that are specific to itself. *See also* type inheritance and table inheritance.

initialize To prepare for execution. To initialize a *variable*, you assign it a starting value. To initialize the database server, you start its operation.

inmigration The process by which Optical Subsystem migrates TEXT and BYTE data from the optical storage subsystem into the Dynamic Server environment.

inner join *See* simple join.

input The information that is received from an external source (for example, from the keyboard, a file, or another program) and processed by a program.

input parameter A placeholder within a prepared SQL statement that indicates a value is to be provided at the time the statement is executed.

insert cursor A cursor for insert operations, associated with an INSERT statement. Allows bulk insert data to be buffered in memory and written to disk.

installation The loading of software from some magnetic medium (tape, cartridge, or floppy disk) or CD onto a computer and preparing it for use.

interactive Refers to a program that accepts input from the user, processes the input, and then produces output on the screen, in a file, or on a printer.

internationalization The process of making Informix products easily adaptable to the conventions of various cultures and natural languages. Among other features, internationalized software can support culturally-specific collation and date, time, and money formats. *See also* the *Informix Guide to GLS Functionality*.

interquery parallelization The ability to process multiple queries simultaneously to avoid a performance delay when multiple independent queries access the same table. *See also* intraquery parallelization.

interrupt A signal from a user or another process that can stop the current process temporarily or permanently. *See also* signal.

interrupt key A key used to cancel or abort a program or to leave a current menu and return to the menu one level above. On many systems, the interrupt key is CONTROL-C; on some others, the interrupt key is DEL or CONTROL-Break.

intraquery parallelization Breaking of a single query into subqueries by a database server using PDQ and then processing the subqueries in parallel. Parallel processing of this type has important implications when each subquery retrieves data from a fragment of a table. Because each partial query operates on a smaller amount of data, the retrieval time is significantly reduced and performance is improved. *See also* interquery parallelization.

IPX/SPX Acronym for Internetwork Packet Exchange / Sequenced Packet Exchange. It refers to the NetWare network protocol by Novell.

ISAM Acronym for *Indexed Sequential Access Method*. This allows you to find information in a specific order, or to find specific items of information quickly through an index. *See also* access method.

ISAM error Operating system or file access error.

ISO Acronym for the *International Standards Organization*. ISO sets worldwide standards for the computer industry, including standards for character input and manipulation, code sets, and SQL syntax.

ISO8859-1 A code set that contains 256 single-byte characters. Characters 0 through 127 are the ASCII characters. Characters 128 through 255 are mostly non-English characters from European languages; for example, é, ñ and ö.

isolation When multiple users attempt to access common data, the level of independence specifically relating to the locking strategy for read-only SQL requests. The various levels of isolation are distinguished primarily by the length of time that shared locks are (or can be) acquired and held. You can set the isolation level with the SET ISOLATION or SET TRANSACTION statement.

iterator function A cursor function that is written in an external language, such as C or Java.

jagged rows A query result in which rows differ in the number and type of columns they contain because the query applies to more than one table in a table hierarchy.

join — The process of combining information from two or more tables based on some common domain of information. Rows from one table are paired with rows from another table when information in the corresponding rows match on the joining criterion. For example, if a **customer_num** column exists in the **customer** and the **orders** tables, you can construct a query that pairs each **customer** row with all the associated **orders** rows based on the common **customer_num**. *See also* Cartesian product and outer join.

join index — A type of generalized-key (GK) index that contains keys that are the result of a query that joins multiple tables.

jukebox — A physical storage device that consists of one or more optical-disc drives, slots that store optical platters when they are not mounted, and a robotic arm that transfers platters between the slots and the drives. A jukebox is also known as an *autochanger.*

kernel — The part of the operating system that controls processes and the allocation of resources.

key — The items of information that are used to locate a row of data. A key defines the pieces of information for which you want to search as well as the order in which you want to process information in a table. For example, you can index the **last_name** column in a **customer** table to find specific customers or to process the customers in alphabetical order (or in reverse alphabetical order) by their last names (when **last_name** serves as the key).

keyword — A word that has a pre-defined meaning in a programming language. For example, the word SELECT is a keyword in SQL.

kilobyte — A unit of storage that consists of 1024 bytes.

Language Supplement — An Informix product that provides the locale files and error messages to support one or more languages. The International Language Supplement supports several European languages. Informix provides separate Language Supplements for several Asian languages.

large object — A data object that is logically stored in a column of a table, but physically stored independently of the column, due to its size. Large objects can be *simple large objects* (TEXT, BYTE) or *smart large objects* (BLOB, CLOB).

leaf node — Index page containing index items and horizontal pointers to other leaf nodes. A database server creates leaf nodes when the root node becomes full.

level of isolation — *See* isolation.

library A group of precompiled *routines* designed to perform tasks that are common to a given kind of application. An application programming interface (API) can include a library of routines that you can call from application programs. *See also* dynamic link library (DLL), shared library, and shared-object file.

light append An unbuffered, unlogged insert operation.

link To combine separately compiled program modules, usually into an executable program. *Compare with* compile and execute.

LIST A *collection data type* created with the LIST constructor in which *elements* are ordered and duplicates are allowed.

literal The representation of a data type value in a format that the database server accepts in data-entry operations. For example, 234 is a literal integer and "abcd" is a literal character.

little-endian A hardware-determined storage method in which the least-significant byte of a multibyte number has the lowest address. *See also* big-endian.

load job The information required to load data into a relational database using the HPL. This information includes the format, map, filter, device array, project, and special options.

local copy For Extended Parallel Server, a replica of a table on a local coserver that is copied to multiple coservers. This allows faster access to the data for OLTP transactions connected to those coservers because you do not have to send the data across the communication links between coservers.

local loopback A connection between the client application and database server that uses a network connection even though the client application and the database server are on the same computer.

local variable A *variable* or *identifier* whose scope of reference is only within the *routine* in which it is defined. *Compare with* global variable.

locale A set of Informix files that specify the linguistic conventions for a country, region, culture, or language. Informix products provide predefined locales that customers cannot modify. A locale provides the name of the code set that the application data uses, the collation order to use for character data, and the end-user format. *See also* client locale, database locale, default locale, server locale, and server-processing locale.

localized order The collation order as specified within a given locale. Localized order can also specify a dictionary or phone-book order. For example, in dictionary order, uppercase characters and lowercase characters are treated the same; one does not take precedence over the other.

locally-detached index For Extended Parallel Server, a type of index that has a fragmentation strategy that is independent of the table fragmentation but where the database server recognizes that each index row resides on the same co-server as the referenced data row. You can use an expression, system-defined hash, or hybrid distribution scheme to create locally detached indexes for any table. *See also* globally-detached index.

lock coupling A locking feature that holds a lock on the child node until a lock is obtained on the parent node during upward movement when updating an R-tree index. Lock coupling is used if the bounding box of a leaf node has changed. You must propagate the change to the parent node by moving upwards in the tree until you reach a parent node that needs no change.

lock mode An option to specify whether a user who requests a lock on an already locked object waits until the object is released to receive the lock, or immediately receives an error, or waits for some span of time before receiving an error.

locking A concurrency feature temporarily limiting access to an object (database, table, page, or row) to prevent conflicting interactions among concurrent processes. An exclusive lock restricts read and write access to only one user; a shared lock allows read-only access to other users. An update lock begins as a shared lock, but is upgraded to an exclusive lock after a row is changed.

locking granularity The size of a locked object. The size can be a database, table, page, or row.

logical log An allocation of disk space that the database server manages that contains records of all changes that were performed on a database during the period when the log was active. It is used to roll back transactions, recover from system failures, and restore databases from archives. *See also* physical log.

login The process of identifying oneself to a computer.

login password *See* Informix user password.

login user ID *See* Informix user ID.

logslice A dbslice whose contents are comprised solely of logical-log files. The logical-log files in the logslice can be owned by multiple coservers, one log file per dbspace. *See also* dbslice, rootslice, and physslice.

LVARCHAR A *built-in data type* that stores varying-length character data of up to 32 kilobytes.

macro A named set of instructions that the computer substitutes when it encounters the name in source code.

mantissa The significant digits in a floating-point number.

map A description of the relation between the records of a data file and the columns of a relational database. *See also* component.

massively parallel processing system A system composed of multiple computers that are connected to a single high-speed communication subsystem. MPP computers can be partitioned into nodes. *Compare with* symmetric multiprocessing system.

megabyte A unit of storage that equals 1024 kilobytes or 1024^2 bytes.

Memory Grant Manager (MGM) (Not for Extended Parallel Server) A database server component that coordinates the use of memory and I/O bandwidth for decision-support queries. MGM uses the DS_MAX_QUERIES, DS_TOTAL_MEMORY, DS_MAX_SCANS, and PDQPRIORITY configuration parameters to determine what resources can or cannot be granted to a decision-support query.

menu A screen display that allows you to choose the commands that you want the computer to perform.

MGM Acronym for Memory Grant Manager.

mirroring Storing the same data on two chunks simultaneously. If one chunk fails, the data is still usable on the other chunk in the mirrored pair. The database server administrator handles this data storage option.

MODE ANSI The keywords specified on the CREATE DATABASE or START DATABASE statement to make a database ANSI compliant.

monochrome A term that describes a monitor that can display only one color.

MPP Acronym for *massively parallel processing system*.

multibyte character
A character that might require from two to four bytes of storage. If a language contains more than 256 characters, the code set must contain multibyte characters. With a multibyte code set, an application cannot assume that one character requires only one byte of storage. *See also* single-byte character.

multiplexed connection
A single network connection between a database server and a client application that handled multiple database connections from the client.

MULTISET
A *collection data type* created with the MULTISET constructor in which *elements* are not ordered and duplicates are allowed.

multithreading
Running of multiple threads within the same process. *See* thread.

named row type
A *row type* created with the CREATE ROW TYPE statement that has a defined name and *inheritance* properties and can be used to construct a *typed table*. A named row type is not equivalent to another named row type, even if its field definitions are the same.

national character
In the context of Native Language Support (NLS), a character in a native language character set. Also known as *native character.*

native character
See national character.

Native Language Support (NLS)
A feature that supports single-byte code sets, using NCHAR and NVARCHAR columns to store non-English character strings. This technology has been superseded on Dynamic Server by Global Language Support (GLS).

NLS
See Native Language Support (NLS).

node
In the context of an index for a database, a node is an ordered group of key values having a fixed number of elements. (A key is a value from a data record.) A B-tree for example, is a set of nodes that contain keys and pointers arranged in a hierarchy. *See also* branch node, leaf node, and root node.

In the context of a MPP system, a node is an individual computer. *See also* massively parallel processing system.

In the context of a SMP system, a node can either be the entire SMP computer or a fully functioning subsystem that uses a portion of the hardware resources of that SMP system. *See also* symmetric multiprocessing system.

For Extended Parallel Server, a node is an individual computer with one or more CPUs that runs a single instance of an operating system within a parallel-processing platform. A node can be a uniprocessor, a cluster of stand-alone computers, an SMP computer, or an independent subsystem configured within an SMP computer.

non-ASCII character A character with a code point greater than 127. Non-ASCII characters include 8-bit characters and multibyte characters.

noncursor function A *user-defined function* that returns a single group of values (one row of data) and therefore does not require a cursor when it is executed. *Compare with* cursor function.

nonvariant function A user-defined function that always returns the same value when passed the same arguments. A nonvariant function must not contain SQL statements. *Compare with* variant function.

not-null constraint A constraint on a column that specifies that the column cannot contain NULL values.

NULL A keyword of SQL, indicating that a value that is unknown, missing, or not applicable. (A NULL value is not the same as a value of zero or blank.)

object *See* database object.

object mode The state of a database object as recorded in the **sysobjstate** system catalog table. A constraint or unique index can be in enabled, disabled, or filtering mode. A trigger or duplicate index can be in enabled or disabled mode. You use SET statements to change the object mode of an object.

object-relational database A database that adds object-oriented features to a *relational database*, including support for *user-defined data types*, *user-defined routines*, user-defined *casts*, user-defined access methods, and *inheritance*.

OLTP Acronym for Online Transaction Processing. *See also* online transaction processing application.

online transaction processing application Characterized by quick, indexed access to a small number of data items. The applications are typically multiuser, and response times are measured in fractions of seconds. See also decision-support application.

online transaction processing queries
The transactions that OLTP applications handle are usually simple and predefined. A typical OLTP system is an order-entry system where only a limited number of rows are accessed by a single transaction many times. *See also* decision-support query.

opaque data type
A fundamental *data type* that you define, which contains one or more values encapsulated with an internal length and input and output functions that convert text to and from an internal storage format. Opaque types need *user-defined routines* and *user-defined operators* that work on them. Synonym for *base type* and *user-defined base type*.

opaque-type support function
One of a group of *user-defined functions* that the database server uses to perform operations on *opaque data types* (such as converting between the internal and external representations of the type).

open
The process of making a resource available, such as preparing a file for access, activating a column, or initiating a window.

operational table
A logging permanent table that uses light appends for fast update operations. Operational tables do not perform record-by-record logging.

operator
In an SQL statement, a symbol (such as =, >, <,+, -, or *) that invokes an *operator function*. The operands to the operator are arguments to the *operator function*.

operator binding
The implicit invocation of an *operator function* when an *operator* is used in an SQL statement.

operator class
An association of *operator-class functions* with a *secondary access method*. The database server uses an operator class to optimize queries and build an index of that secondary access method.

operator-class function
One of the *operator-class support functions* or *operator-class strategy functions* that constitute an *operator class*. For user-defined operator classes, the operator-class functions are *user-defined functions*.

operator-class strategy function
An *operator-class function* that can appear as a *filter* in a query. The query optimizer uses the strategy functions to determine if an *index* of a particular *secondary access method* can be used to process the filter. You register operator-class strategy functions in the STRATEGIES clause of the CREATE OPCLASS statement.

operator-class support function An *operator-class function* that a *secondary* access method uses to build or search an *index*. You register operator-class support functions in the SUPPORT clause of the CREATE OPCLASS statement.

operator function A *function* that processes one or more arguments (its operands) and returns a value. Many operator functions have corresponding operators, such as **plus()** and +. You can overload an operator function so that it handles a *user-defined data type*. *See also* routine overloading.

optical cluster An amount of space on an optical disc that is reserved for storing a group of logically related simple large objects or smart large objects.

optical family A group of optical discs, theoretically unlimited in number.

optical platter A removable optical disc that stores data in an optical storage subsystem.

optical statements The SQL statements that you use to control optical clustering.

optical volume One side of a removable Write-Once-Read-Many (WORM) optical disc.

outer join An asymmetric joining of a dominant (*outer*) table and a subservient table in a query where the values for the outer part of the join are preserved even though no matching rows exist in the subservient table. Any dominant-table rows that do not have a matching row in the subservient table contain NULL values in the columns selected from the subservient table.

outmigration The process by which Optical Subsystem migrates TEXT or BYTE data from the Dynamic Server environment to an optical storage subsystem.

output The result that the computer produces in response to a query or a request for a report.

overloading *See* routine overloading.

owner-privileged A class of SPL routines that any user can create who has Resource database privileges.

packed decimal A storage format that represents either two decimal digits or a sign and one decimal digit in each byte.

pad Usually, to fill empty places at the beginning or end of a line, string, or field with spaces or blanks when the input is shorter than the field.

page The physical unit of disk storage and basic unit of memory storage that the database server uses to read from and write to Informix databases. Page size is fixed for a particular operating system and platform. A page is always entirely contained within a chunk. *See also* home page and remainder page.

parallel database query The execution of SQL queries in parallel rather than sequential order. The tasks a query requires are distributed across several processors. This type of distribution enhances database performance.

parallel-processing platform A parallel-processing platform is a set of independent computers that operate in parallel and communicate over a high-speed network, bus, or interconnect. *See also* symmetric multiprocessing system and massively parallel processing system.

parallelism Ability of an Informix database server to process a task in parallel by breaking the task into subtasks and processing the subtasks simultaneously, thus improving performance.

parameter A variable that is given a value for a specified application. In the signature of a user-defined routine, a parameter serves as a placeholder for an argument. The parameter specifies the *data type* of the value that the user-defined routine expects when it receives the associated argument at runtime. *See also* configuration file, input parameter, and routine signature.

parent-child relationship *See* referential constraint.

parent table The referenced table in a referential constraint. *See also* child table.

participating coserver A coserver that controls one or more fragments of a table that Extended Parallel Server accesses. *See also* coserver and connection coserver.

partition *See* table fragment.

pattern An identifiable or repeatable series of characters or symbols.

PDQ Acronym for *parallel database query.*

PDQ priority Determines the amount of resources that a database server allocates to process a query in parallel. These resources include memory, threads (such as scan threads), and sort space. The level of parallelism is established by using the **PDQPRIORITY** environment variable or various database server configuration parameters (including PDQPRIORITY and MAX_PDQPRIORITY) or dynamically through the SET PDQPRIORITY statement.

permission On some operating systems, the right to access files and directories.

phantom row A row of a table that is initially modified or inserted during a transaction but is subsequently rolled back. Another user can see a phantom row if the isolation level is Informix Dirty Read or ANSI compliant Read Uncommitted. No other isolation level allows a user to see a changed but uncommitted row.

physical log A set of contiguous disk pages in shared memory where the database server stores an unmodified copy (before-image) of pages before the changed pages are recorded. The pages in the physical log can be any database server page except a blobspace blobpage.

physslice A dbslice that contains the physical log. *See also* dbslice, logslice, and rootslice.

pointer A value that specifies the address in memory of the data or *variable*, rather than the contents of the data or variable.

polymorphism *See* routine overloading and type substitutability.

precision The total number of significant digits in a real number, both to the right and left of the decimal point. For example, the number 1437.2305 has a precision of 8. *See also* scale.

predefined opaque data type An opaque data type for which the database server provides the type definition. *See also* BLOB, BOOLEAN, CLOB, LVARCHAR and pointer.

predicate *See* filter.

predicate lock A lock held on index keys that qualifies for a predicate. In a predicate lock, exclusive predicates consist of a single key value, and shared predicates consist of a query rectangle and a scan operation such as inclusion or overlap.

prepared statement An SQL statement that is generated by the PREPARE statement from a character string or from a variable that contains a character string. This feature allows you to form your request while the program is executing without having to modify and recompile the program.

preprocessor A program that takes high-level programs and produces code that a standard language compiler such as C can compile.

primary access method An access method whose *routines* access a *table* with such operations as inserting, deleting, updating, and scanning. *See also* secondary access method.

primary key The information from a column or set of columns that uniquely identifies each row in a table. The primary key sometimes is called a *unique key.*

primary-key constraint Specifies that each entry in a column or set of columns contains a unique non-NULL value.

printable character A character that can be displayed on a terminal, screen, or printer. Printable characters include A-Z, a-z, 0-9, and punctuation marks. *Compare with* control character.

privilege The right to use or change the contents of a database, table, table fragment, or column. *See also* access privileges.

procedure A *routine* that does not return values. *See also* user-defined procedure.

procedure overloading *See* routine overloading.

process A discrete task, generally a program, that the operating system executes.

project A group of related components that the High-Performance Loader (HPL) uses. *See also* component.

projection Taking a subset from the columns of a single table. Projection is implemented through the select list in the SELECT clause of a SELECT statement and returns some of the columns and all the rows in a table. *See also* selection and join.

promotable lock A lock that can be changed from a shared lock to an exclusive lock. *See also* update lock.

protocol A set of conventions that govern communication among computers. These conventions govern format, timing, sequencing, and error control.

query A request to the database to retrieve data that meet certain criteria, usually made with the SELECT statement.

query optimization information statements The SQL statements that are used to optimize the performance of queries. These statements include SET EXPLAIN, SET OPTIMIZATION, and UPDATE STATISTICS.

query unnesting An execution strategy for nested SQL subqueries whereby Extended Parallel Server rewrites such subqueries to use modified joins rather than iteration mechanisms. The **sqexplain.out** file reflects the query plan that has been selected after subquery unnesting has occurred.

R-tree index (Not for Extended Parallel Server) A type of index that uses a tree structure based on overlapping bounding rectangles to speed access to spatial and multidimensional data types. *See also* bitmap index and B-tree index.

range fragmentation A distribution scheme that distributes data in table fragments that contain a specified key range. This technique can eliminate scans of table fragments that do not contain the required rows, making queries faster.

range rule A user-defined algorithm for *expression-based fragmentation*. It defines the boundaries of each fragment in a table using SQL relational and logical operators. Expressions in a range rule can use the following restricted set of operators: >, <, >=, <=, and the logical operator AND.

raw device *See* unbuffered disk I/O.

raw disk *See* unbuffered disk I/O.

raw table A nonlogged permanent table that uses *light appends*.

Read Committed An level of isolation that the SET TRANSACTION statement can specify, in which a user can view rows that are currently committed at the moment of the query request, but cannot view rows that were changed as part of a currently uncommitted transaction. This is the default isolation level for databases that are not ANSI compliant. *See also* Committed Read.

Read Uncommitted An ANSI compliant level of isolation, set with the SET TRANSACTION statement, that does not account for locks. This allows a user to view any existing rows, even rows that currently can be altered within currently uncommitted transactions. Read Uncommitted is the lowest level of isolation (no isolation at all), and is thus the most efficient. *See also* Dirty Read.

real user ID *See* Informix user ID.

record *See* row.

Record-ID A four-byte RSAM entity, also known as RID, that describes the logical position of the record within a fragment. Not the same as rowid.

recover a database To restore a database to a former condition after a system failure or other destructive event. The recovery restores the database as it existed immediately before the failure.

referential constraint The relationship between columns within a table or between tables; also known as a *parent-child relationship*. Referencing columns are also known as *foreign keys*.

registering In a database, the process of storing information about a *database object* in the *system catalog tables* of a database. Most SQL data definition statements perform some type of registration. For example, the CREATE FUNCTION and CREATE PROCEDURE statements register a *user-defined routine* in a database.

relation *See* table.

relational database A database that uses table structures to store data. Data in a relational database is divided across tables in such a way that additions and modifications to the data can be made easily without loss of information.

relational database server A database server that manages data values that are stored in rows and columns.

remainder page A page that accommodates subsequent bytes of a long data row. If the trailing portion of a data row is less than a full page, it is stored on a remainder page. After the database server creates a remainder page for a long row, it can use the remaining space in the page to store other rows. Each full page that follows the home page is referred to as a big-remainder page.

remote connection A connection that requires a network.

remote routine A routine in a databases of a remote server. *See* subordinate server.

remote server *See* subordinate server.

remote table In a distributed query, a table in a database of a server that is not the local database server. *See also* coordinating server, subordinate server.

Repeatable Read An Informix and ANSI level of isolation available with the Informix SET ISOLATION statement or the ANSI compliant SET TRANSACTION statement, which ensures that all data values read during a transaction are not modified during the entire transaction. Transactions under ANSI Repeatable Read are also known as Serializable. Informix Repeatable Read is the default level of isolation for ANSI compliant databases. *See also* isolation and Serializable.

reserved pages The first 12 pages of the initial chunk of the root dbspace. Each reserved page stores specific control and tracking information that the database server uses.

reserved word A word in a statement or command that you cannot use in any other context of the language or program without receiving a warning or error message.

restore a database *See* recover a database.

role A classification or work task, such as **payroll**, that the DBA assigns. Assignment of roles makes management of privileges convenient.

role separation (Not for Extended Parallel Server) A database server installation option that allows different users to perform different administrative tasks.

roll back The process that reverses an action or series of actions on a database. The database is returned to the condition that existed before the actions were executed. *See also* transaction and commit work.

root dbspace The initial *dbspace* that the database server creates. It contains reserved pages and internal tables that describe and track all other dbspaces, blobspaces, sbspaces, tblspaces, chunks, and databases.

root node A single index page that contains node pointers to *branch nodes*. The database server allocates the root node when you create an index for an empty table.

root supertype The *named row type* at the top of a *type hierarchy.* A root supertype has no *supertype* above it.

rootslice A dbslice that contains the root dbspaces for all coservers for Extended Parallel Server. *See also* dbslice, logslice, and physslice.

round-robin fragmentation A distribution scheme in which the database server distributes rows sequentially and evenly across specified dbspaces.

routine A group of program statements that perform a particular task. A routine can be a *function* or a *procedure.* All routines can accept arguments. *See also* built-in and user-defined routine.

routine modifier A keyword in the WITH clause of a CREATE FUNCTION, CREATE PROCEDURE, ALTER FUNCTION, ALTER PROCEDURE, or ALTER ROUTINE statement that specifies a particular attribute or usage of a *user-defined routine.*

routine overloading The ability to assign one name to multiple *user-defined routines* and specify *parameters* of different data types on which each routine can operate. An overloaded routine is uniquely defined by its *routine signature.*

routine resolution The process that the database server uses to determine which *user-defined routine* to execute, based on the *routine signature. See also* routine overloading.

routine signature The information that the database server uses to uniquely identify a *user-defined routine*. The signature includes the type of routine (function or procedure); the routine name; and the number, order, and data types of the parameters. *See also* routine overloading and specific name.

row A group of related items of information about a single entity across all columns in a database table. In a table of customer information, for example, a row contains information about a single customer. A row is sometimes referred to as a *record* or *tuple*. In an object-relational model, each row of a table stands for one *instance* of the subject of the table, which is one particular example of that entity. In a screen form, a row can refer to a line of the screen.

row type A *complex data type* that contains one or more related data *fields*, of any *data type*, that form a template for a record. The data in a row type can be stored in a row or column. *See also* named row type and unnamed row type.

row variable An Informix ESQL/C *host variable* or *SPL variable* that holds an entire *row type* and provides access to the individual *fields* of the row.

rowid In nonfragmented tables, rowid refers to an integer that defines the physical location of a row. Rowids must be explicitly created to be used in fragmented tables and they do not define a physical location for a row. Rowids in fragmented tables are accessed by an index that is created when the rowid is created; this access method is slow. Informix recommends that users creating new applications move toward using primary keys as a method of row identification instead of using rowids.

rule How a database server or a user determines into which fragment rows are placed. The database server determines the rule for *round-robin fragmentation* and *system-defined hash fragmentation*. The user determines the rule for *expression-based fragmentation* and *hybrid fragmentation*. *See also* arbitrary rule and range rule.

runtime environment The hardware and operating-system services available at the time a program runs.

runtime error An error that occurs during program execution. *Compare with* compile-time error.

sbspace (Not for Extended Parallel Server) A logical storage area that contains one or more chunks that store only BLOB and CLOB data.

scale The number of digits to the right of the decimal place in DECIMAL notation. The number 14.2350 has a scale of 4 (four digits to the right of the decimal point). *See also* precision.

scale up The ability to compensate for an increase in query size by adding a corresponding amount of computer resources so that processing time does not also increase.

scan thread A database server thread that is assigned the task of reading rows from a table. When a query is executed in parallel, the database server allocates multiple scan threads to perform the query in parallel.

schema The structure of a database or a table. The schema for a table lists the names of the columns, their data types, and (where applicable) the lengths, indexing, and other information about the structure of the table.

scope of reference The portion of a *routine* or application program where an *identifier* can be accessed. Three possible scopes exist: local (applies only in a single statement block), modular (applies throughout a single module), and global (applies throughout the entire program). *See also* local variable and global variable.

scroll cursor A cursor created with the SCROLL keyword that allows you to fetch rows of the active set in any sequence.

secondary access method An access method whose *routines* access an *index* with such operations as inserting, deleting, updating, and scanning. *See also* operator class and primary access method.

secure auditing (Not for Extended Parallel Server) A facility of Informix database servers that lets a database server administrator keep track of unusual or potentially harmful user activity. Use the **onaudit** utility to enable auditing of events and create audit masks, and the **onshowaudit** utility to extract the audit event information for analysis.

select *See* query.

select cursor A cursor that is associated with a SELECT statement, which lets you scan multiple rows of data, moving data row by row into a set of receiving variables.

selection Taking a horizontal subset of the rows of a single table that satisfies a specified condition. Selection is implemented through the WHERE clause of a SELECT statement and returns some of the rows and all of the columns in a table. *See also* projection and join.

selective index A type of *generalized-key index* that contains keys for only a subset of a table.

selectivity The proportion of rows within the table that a query filter can pass.

self-join A join between a table and itself. A self-join occurs when a table is used two or more times in a SELECT statement (with different aliases) and joined to itself.

semaphore An operating-system communication device that signals a process to awaken.

sequential cursor A cursor that can fetch only the next row in sequence. A sequential cursor can read through a table only once each time the sequential cursor is opened.

Serializable An ANSI compliant level of isolation set with the SET TRANSACTION statement, ensuring all data read during a transaction is not modified during the entire transaction. *See also* isolation and Repeatable Read.

server locale The locale that a database server uses when it performs its own read and write operations. The **SERVER_LOCALE** environment variable can specify a nondefault locale. *See also* client locale and locale.

server name The unique name of a database server, assigned by the database server administrator, that an application uses to select a database server.

server number A unique number between 0 and 255, inclusive, that a database server administrator assigns when a database server is initialized.

server-processing locale The locale that a database server determines dynamically for a given connection between a client application and a database. *See also* locale.

session The structure that is created for an application using the database server.

SET A *collection data type* created with the SET type constructor, in which *elements* are not ordered and duplicate values can be inserted.

shared library A *shared-object file* on a UNIX system. *See also* dynamic link library (DLL).

shared lock A lock that more than one thread can acquire on the same object. Shared locks allow for greater concurrency with multiple users; if two users have shared locks on a row, a third user cannot change the contents of that row until both users (not just the first) release the lock. Shared-locking strategies are used in all levels of isolation except Informix Dirty Read and ANSI compliant Read Uncommitted.

shared memory A portion of main memory that is accessible to multiple processes. Shared memory allows multiple processes to communicate and access a common data space in memory. Common data does not have to be reread from disk for each process, reducing disk I/O and improving performance. Also used as an Inter-Process Communication (IPC) mechanism to communicate between two processes running on the same computer.

shared-object file A *library* that is not linked to an application at compile time but instead is loaded into memory by the operating system as needed. Several applications can share access to the loaded shared-object file. *See also* dynamic link library (DLL) *and* shared library.

shelf The location of an optical platter that is neither on an optical drive nor in a jukebox slot.

shuffling Shuffling refers to the process that occurs when a database server moves rows or key values from one fragment to another. Shuffling occurs in a variety of circumstances including when you attach, detach, or drop a fragment.

signal A means of asynchronous communication between two processes. For example, signals are sent when a user or a program attempts to interrupt or suspend the execution of a process.

signature *See* routine signature.

simple join A join that combines information from two or more tables based on the relationship between one column in each table. Rows that do not satisfy the join criteria are discarded from the result. Also known as an inner join. *See also* composite join.

simple large object A *large object* that is stored in a *blobspace* or *dbspace* is not recoverable and does not obey transaction isolation modes. Simple large objects include TEXT and BYTE data types. Extended Parallel Server does not support simple large objects that are stored in a blobspace.

simple predicate A search condition in the WHERE clause that has one of the following forms: **f(column, constant)**, **f(constant, column)**, or **f(column)**, where **f** is a binary or unary function that returns a Boolean value (TRUE, FALSE, or UNKNOWN).

single-byte character A character that uses one byte of storage. Because a single byte can store values in the range of 0 to 255, it can uniquely identify 256 characters. With these code sets, an application can assume that one character is always stored in one byte. *See also* 8-bit character and multibyte character.

singleton implicit transaction A single-statement transaction that does not require either a BEGIN WORK or a COMMIT WORK statement. This type of transaction can occur only in a database that is not ANSI compliant, but that supports transaction logging. *See also* explicit transaction and implicit transaction.

singleton select A SELECT statement that returns a single row.

smart large object A *large object* that is stored in an *sbspace*, which has read, write, and seek properties similar to a UNIX file, is recoverable, obeys transaction isolation modes, and can be retrieved in segments by an application. Smart large objects include BLOB and CLOB data types.

SMI Acronym for *system-monitoring interface*.

SMP *See* symmetric multiprocessing system.

source file A text file that contains instructions in a high-level language, such as C. A C source file is *compiled* into an *executable file* called an object file. An SPL source file is compiled into its own executable format. *See also* compile.

source type The data type from which a DISTINCT type is derived.

specific name A name that you can assign to an overloaded *user-defined routine* to uniquely identify a particular signature of the user-defined routine. *See also* routine overloading and routine signature.

speed up The ability to add computing hardware to achieve correspondingly faster performance for a DSS query or OLTP operation of a given volume.

SPL *See* Stored Procedure Language (SPL).

SPL function An *SPL routine* that returns one or more values.

SPL procedure An *SPL routine* that does not return a value.

SPL routine A *user-defined routine* that is written in Stored Procedure Language (*SPL*). Its name, parameters, executable format, and other information are stored in the system catalog tables of a database. An SPL routine can be an *SPL procedure* or an *SPL function*.

SPL variable A *variable* that is declared with the DEFINE statement in an *SPL routine*.

SQL Acronym for *Structured Query Language*. SQL is a database query language that was developed by IBM and standardized by ANSI. Informix relational database management products are based on an extended implementation of ANSI-standard SQL.

SQL API An *application programming interface* that allows you to embed SQL statements directly in an application. The embedded-language product Informix ESQL/C is an example of an SQL API. *See also* host variable.

SQLCA Acronym for *SQL Communications Area*. The SQLCA is a data structure that stores information about the most recently executed SQL statement. The result code returned by the database server to the SQLCA is used for error handling by Informix SQL APIs.

sqlda Acronym for *SQL descriptor area*. A dynamic SQL management structure that can be used with the DESCRIBE statement to store information about database columns or host variables used in dynamic SQL statements. The **sqlda** structure is an Informix-specific structure for handling dynamic columns. It is available only within an Informix ESQL/C program. *See also* descriptor and system-descriptor area.

sqlhosts A file that identifies the types of connections the database server supports.

stack operator Operators that allow programs to manipulate values that are on the stack.

staging-area blobspace (Not for Extended Parallel Server) The blobspace where a database server temporarily stores TEXT or BYTE data that is being outmigrated to an optical storage subsystem.

statement A line or set of lines of program code that describes a single action (for example, a SELECT statement or an UPDATE statement).

statement block A unit of SPL program code that performs a particular task and is usually marked by the keywords BEGIN and END. The statement block of an *SPL routine* is the smallest scope of reference for program variables.

statement identifier An embedded variable name or SQL statement identifier that represents a data structure defined in a PREPARE statement. It is used for dynamic SQL statement management by Informix SQL APIs.

static table A nonlogging, read-only permanent table.

status variable A program variable that indicates the status of some aspect of program execution. Status variables often store error numbers or act as flags to indicate that an error has occurred.

storage space A *dbspace*, *blobspace*, or *sbspace* that is used to hold data.

stored procedure A legacy term for an *SPL routine*.

Stored Procedure Language (SPL) An Informix extension to SQL that provides flow-control features such as sequencing, branching, and looping. *See also* SPL routine.

strategy function *See* operator-class strategy function.

string A sequence of characters (typically alphanumeric) that is manipulated as a single unit. A string might consist of a word (such as 'Smith'), a set of digits representing a number (such as '19543'), or any other collection of characters. Strings generally are delimited by single quotes. *String* is also a character data type, available in Informix ESQL/C programs, in which the character string is stripped of trailing blanks and is null terminated.

subordinate server Any database server in a distributed query that did not initiate the query. Sometimes called remote server. *See also* coordinating server.

subordinate table *See* outer join.

subquery A query that is embedded as part of another SQL statement. For example, an INSERT statement can contain a subquery in which a SELECT statement supplies the inserted values in place of a VALUES clause; an UPDATE statement can contain a subquery in which a SELECT statement supplies the updating values; or a SELECT statement can contain a subquery in which a second SELECT statement supplies the qualifying conditions of a WHERE clause for the first SELECT statement. (Parentheses always delimit a subquery, unless you are referring to a CREATE VIEW statement or unions.) Subqueries are always parallelized. *See also* correlated subquery and independent subquery.

subscript A subscript is an integer-valued offset into an array. Subscripts can be used to indicate the start or end position in a character data-type variable.

substring A portion of a character string.

subtable A *typed table* that inherits properties (column definitions, constraints, triggers) from a *supertable* above it in the *table hierarchy* and can add additional properties.

subtype A *named row type* that inherits all representation (data *fields*) and behavior (*routines*) from a *supertype* above it in the *type hierarchy* and can add additional fields and routines. The number of fields in a subtype is always greater than or equal to the number of fields in its supertype.

supertable A *typed table* whose properties (constraints, storage options, triggers) are inherited by a *subtable* beneath it in the *table hierarchy*. The scope of a query on a supertable is the supertable and its subtables.

supertype A *named row type* whose representation (data *fields*) and behavior (*routines*) is inherited by a *subtype* below it in the *type hierarchy*.

support function *See* aggregate support function, opaque-type support function, and operator-class support function.

support routine *See* support function.

symmetric multiprocessing system A system composed of multiple computers that are connected to a single high-speed communication subsystem. An SMP has fewer computers than an MPP system and cannot be partitioned into nodes. *Compare with* massively parallel processing system.

synonym A name that is assigned to a table, view, or sequence, and that can be used in place of the original name. A synonym does not replace the original name; instead, it acts as an alias for the table, view, or sequence.

sysmaster database A database on each database server that holds the ON-Archive catalog tables and *system-monitoring interface* (SMI) tables that contain information about the state of the database server. The database server creates the **sysmaster** database when it initializes disk space.

system call A routine in an operating-system *library* that programs call to obtain information from the operating system.

system catalog A group of database tables that contain information about the database itself, such as the names of tables or columns in the database, the number of rows in a table, the information about indexes and database privileges, and so on. *See also* data dictionary.

system-defined cast A pre-defined *cast* that is known to the database server. Each built-in cast performs automatic conversion between two different *built-in data types*.

system-defined hash fragmentation An Extended Parallel Server-defined distribution scheme that maps each row in a table to a set of integers and uses a system-defined algorithm to distribute data evenly by hashing a specified key.

system-descriptor area A dynamic SQL management structure that is used with the ALLOCATE DESCRIPTOR, DEALLOCATE DESCRIPTOR, DESCRIBE, GET DESCRIPTOR, and SET DESCRIPTOR statements to store information about database columns or host variables used in dynamic SQL statements. The structure contains an item descriptor for each column; each item descriptor provides information such as the name, data type, length, scale, and precision of the column. The system-descriptor area is the X/Open standard for handling dynamic columns. *See also* descriptor and sqlda.

system-monitoring interface
A collection of tables and pseudo-tables in the **sysmaster** database that maintains dynamically updated information about the operation of the database server. The tables are constructed in memory but are not recorded on disk. Users can query the SMI tables with the SELECT statement of SQL.

table
A rectangular array of data in which each row describes a single entity and each column contains the values for each category of description. For example, a table can contain the names and addresses of customers. Each row corresponds to a different customer and the columns correspond to the name and address items. A table is sometimes referred to as a *base table* to distinguish it from the views, indexes, and other objects defined on the underlying table or associated with it.

table fragment
Zero or more rows that are grouped together and stored in a dbspace that you specify when you create the fragment. A virtual table fragment might reside in an *sbspace* or an *extspace*.

table fragmentation
A method of separating a table into potentially balanced fragments to distribute the workload and optimize the efficiency of the database operations. Also known as data partitioning. Table-fragmentation methods (also known as distribution schemes) include *expression-based*, *hybrid*, *range*, *round-robin*, and *system-defined hash*.

table hierarchy
A relationship you can define among *typed tables* in which *subtables* inherit the behavior (constraints, triggers, storage options) from *supertables*. Subtables can add additional constraint definitions, storage options, and triggers.

table inheritance
The property that allows a *typed table* to inherit the behavior (constraints, storage options, triggers) from a *typed table* above it in the *table hierarchy*.

target table
The underlying base table that a violations table and diagnostics table are associated with. You use the START VIOLATIONS TABLE statement to create the association between the target table and the violations and diagnostics tables.

tblspace
The logical collection of *extents* that are assigned to a table. It contains all the disk space that is allocated to a given table or table fragment and includes pages allocated to data and to indexes, pages that store TEXT or BYTE data in the dbspace, and bitmap pages that track page use within the extents.

TCP/IP
The specific name of a particular standard transport layer protocol (TCP) and network layer protocol (IP). A popular network protocol used in DOS, UNIX, and other environments.

temporary An attribute of any file, index, or table that is used only during a single session. Temporary files or resources are typically removed or freed when program execution terminates or an online session ends.

TEXT A *data type* for a *simple large object* that stores text and can be as large as 2^{31} bytes. *See also* BYTE.

thread A piece of work or task for a *virtual processor* just as a virtual processor is a task for a CPU. A virtual processor is a task that the operating system schedules for execution on the CPU; a database server thread is a task that a virtual processor schedules internally for processing. Threads are sometimes called *lightweight processes* because they are like processes but make fewer demands on the operating system. *See also* multithreading and user thread.

TLI Acronym for Transport Layer Interface. It is the interface designed for use by application programs that are independent of a network protocol.

trace To keep a running list of the values of program variables, arguments, expressions, and so on, in a program or SPL routine.

transaction A collection of one or more SQL statements that is treated as a single unit of work. If one statement in a transaction fails, the entire transaction can be *rolled back* (canceled). If the transaction is successful, the work is *committed* and all changes to the database from the transaction are accepted. *See also* explicit transaction, implicit transaction, and singleton implicit transaction.

transaction lock A lock on an *R-tree index* that is obtained at the beginning of a transaction and held until the end of the transaction.

transaction logging The process of keeping records of transactions. *See also* logical log.

transaction mode The method by which constraints are checked during transactions. You use the SET statement to specify whether constraints are checked at the end of each data manipulation statement or after the transaction is committed.

trigger A mechanism that resides in the database. It specifies that when a particular action (insert, delete, or update) occurs on a particular table, the database server should automatically perform one or more additional actions.

tuple *See* row.

two-phase commit A protocol that ensures that transactions are uniformly committed or rolled back across multiple database servers. It governs the order in which commit transactions are performed and provides a recovery mechanism in case a transaction does not execute. *See also* heterogeneous commit.

type constructor An SQL keyword that indicates to the database server the type of complex data to create (for example, *LIST*, *MULTISET*, *ROW*, *SET*).

type hierarchy A relationship that you define among *named row types* in which *subtypes* inherit representation (data *fields*) and behavior (*routines*) from *supertypes* and can add more fields and routines.

type inheritance The property that allows a *named row type* or *typed table* to inherit representation (data *fields*, columns) and behavior (*routines*, operators, rules) from a named row type above it in the *type hierarchy*.

type substitutability The ability to use an instance of a subtype when an instance of its supertype is expected.

typed collection variable An ESQL/C *collection variable* or *SPL variable* that has a defined *collection data type* associated with it and can only hold a *collection* of its defined type. *See also* untyped collection variable.

typed table A table that is constructed from a *named row type* and whose rows contain instances of that *row type*. A typed table can be used as part of a *table hierarchy*. The columns of a typed table correspond to the *fields* of the named row type.

UDA *See* user-defined aggregate.

UDR *See* user-defined routine.

UDT *See* user-defined data type.

unbuffered disk I/O Disk I/O that is controlled directly by the database server instead of the operating system. This direct control helps improve performance and reliability for updates to database data. Unbuffered I/O is supported by character-special files on UNIX and by both unbuffered files and the raw disk interface on Windows NT.

Uncommitted Read *See* Read Uncommitted.

uncorrelated subquery *See* independent subquery.

unique constraint Specifies that each entry in a column or set of columns has a unique value.

unique index An index that prevents duplicate values in the indexed column.

unique key *See* primary key.

UNIX real user ID *See* Informix user ID.

unload job The information required to unload data from a relational database using the HPL. This information includes format, map, query, device array, project, and special options.

unlock To free an object (database, table, page, or row) that has been locked. For example, a locked table prevents others from adding, removing, updating, or (in the case of an exclusive lock) viewing rows in that table as long as it is locked. When the user or program unlocks the table, others are permitted access again.

unnamed row type A *row type* created with the ROW constructor that has no defined name and no inheritance properties. Two unnamed row types are equivalent if they have the same number of *fields* and if corresponding fields have the same *data type*, even if the fields have different names.

untyped collection variable A generic ESQL/C *collection variable* or *SPL variable* that can hold a *collection* of any *collection data type* and takes on the data type of the last collection assigned to it. *See also* typed collection variable.

update The process of changing the contents of one or more columns in one or more existing rows of a table.

update lock A promotable lock that is acquired during a SELECT...FOR UPDATE. An update lock behaves like a shared lock until the update actually occurs, and it then becomes an exclusive lock. It differs from a shared lock in that only one update lock can be acquired on an object at a time.

user-defined aggregate An aggregate function that is not provided by the database server (built in) that includes extensions to built-in aggregates and newly defined aggregates. The database server manages all aggregates.

user-defined base type *See* opaque data type.

user-defined cast A *cast* that a user creates with the CREATE CAST statement. A user-defined cast typically requires a *cast function*. A user-defined cast can be an *explicit cast* or an *implicit cast*.

user-defined data type A *data type that you define for use in a relational database. You can define opaque data types and distinct data types.*

user-defined function A *user-defined routine* that returns at least one value. You can write a user-defined function in SPL (*SPL function*) or in an external language that the database server supports (*external function*).

user-defined procedure A *user-defined routine* that does not return a value. You can write a user-defined procedure in SPL (*SPL procedure*) or in an external language that the database server supports (*external procedure*).

user-defined routine A *routine* that you write and register in the system catalog tables of a database, and that an SQL statement or another routine can invoke. You can write a user-defined routine in SPL (*SPL routine*) or in an external language (*external routine*) that the database server supports.

user ID *See* Informix user ID.

user ID password *See* Informix user password.

user name *See* Informix user ID.

user password *See* Informix user password.

user thread User threads include session threads (called **sqlexec** threads) that are the primary threads that the database server runs to service client applications. User threads also include a thread to service requests from the **onmode** utility, threads for recovery, and page-cleaner threads. *See* thread.

variable The *identifier* for a location in memory that stores the value of a program object whose value can change during program execution. *Compare with* constant, macro, and pointer.

variant function A *user-defined function* that might return different values when passed the same arguments. A variant function can contain SQL statements. *Compare with* nonvariant function.

view A dynamically controlled picture of the contents in a database that allows a programmer to determine what information the user sees and manipulates. A view represents a virtual table based on a specified SELECT statement.

violations table A special table that holds rows that fail to satisfy constraints and unique index requirements during data manipulation operations on base tables. You use the START VIOLATIONS TABLE statement to create a violations table and associate it with a base table.

virtual column A derived column of information, created with an SQL statement, that is not stored in the database. For example, you can create virtual columns in a SELECT statement by arithmetically manipulating a single column, such as multiplying existing values by a constant, or by combining multiple columns, such as adding the values from two columns.

virtual-column index A type of *generalized-key index* that contains keys that are the result of an expression.

virtual processor A multithreaded process that makes up the database server and is similar to the hardware processors in the computer. It can serve multiple clients and, where necessary, run multiple threads to work in parallel for a single query.

virtual table A table whose data you create to access data in an external file, external DBMS, or smart large object. The database server does not manage external data or directly manipulate data within a smart large object. The Virtual-Table Interface allows users to access the external data in a virtual table using SQL DML statements and join the external data with Dynamic Server table data.

VLDB Acronym for very large database.

warning A message or other indicator about a condition that software (such as the database server or compiler) detects. A condition that results in a warning does not necessarily affect the ability of the code to run. *See also* compile-time error *and* runtime error.

white space A series of one or more space characters. The GLS locale defines the characters that are considered to be space characters. For example, both the TAB and blank might be defined as space characters in one locale, but certain combinations of the CTRL key and another character might be defined as space characters in a different locale.

wide character A form of a code set that involves normalizing the size of each multibyte character so that each character is the same size. This size must be equal to or greater than the largest character that an operating system can support, and it must match the size of an integer data type that the C compiler can scale. Some examples of an integer data type that the C compiler can scale are short integer (**short int**), integer (**int**), or long integer (**long int**).

wildcard A special symbol representing any character or any string of zero or more characters. In SQL, for example, you can use the asterisk (*), question mark (?), percent sign (%), and underscore (_) as wildcard characters in some contexts. (Asterisk and question mark are also UNIX wildcards.)

window A rectangular area on the screen in which you can take actions without leaving the context of the background program.

WORM Acronym for *Write-Once-Read-Many* optical media. When a bit of data is written to a WORM platter, a permanent mark is made on that optical platter.

X/Open An independent consortium that produces and develops specifications and standards for open-systems products and technology, such as dynamic SQL.

X/Open Portability Guide (XPG) A set of specifications that vendors and users can use to build portable software. Any vendor that carries the XPG brand on a given software product is guaranteeing that the software correctly implements the X/Open Common Applications Environment (CAE) specifications. There are CAE specifications for SQL, XA, ISAM, RDA, and so on.

zoned decimal A data representation that uses the low-order four bits of each byte to designate a decimal digit (0 through 9) and the high-order four bits to designate the sign of the digit.

Index

Numerics

9.3 features, overview 6

A

Abbreviated year values 2-19, 3-28, 3-31, 3-33, 3-50
ACCESS keyword 1-18, 2-50
Access method
B-tree 1-21, 1-47, 3-53
built-in 1-18, 1-21
primary 1-19, 1-67
R-tree 3-53
secondary 1-18, 1-34, 1-49, 2-34
sysams data 1-18
sysindices data 1-49
sysopclasses data 1-53
systabamdata data 1-67
Access privilege. *See* Privilege.
Activity-log files 3-77
AC_CONFIG environment variable 3-24
ac_config.std file 3-24
Addition (+) operator 2-52, 2-70
Administrative listener port 3-61
AFCRASH configuration parameter 3-24
AFDEBUG environment variable 3-24
Aggregate function
built-in 2-30, 2-33, 2-42
no BYTE argument 2-11
no collection arguments 2-30, 2-33, 2-42
no TEXT argument 2-44
sysaggregates data 1-17
user-defined (UDA) 1-17
AIX operating system 3-79
Alias of a table 1-9
Alignment of data type 1-23, 1-79
ALL operator 2-70
ALTER OPTICAL CLUSTER statement 1-55
Alter privilege 1-10, 1-68, 1-82
ALTER TABLE statement
casting effects 2-63
changing data types 2-6
lock mode 3-57
next extent size 1-12
SERIAL columns 2-39
SERIAL8 columns 2-40
systables.version 1-71
am_beginscan( ) function 1-20
am_close( ) function 1-20
am_getnext( ) function 1-19
am_insert( ) function 1-20
am_open( ) function 1-19
AND operator 1-24, 2-70
ANSI compliance
-ansi flag 3-27
DATETIME literals 3-51
DBANSIWARN environment variable 3-27
DECIMAL range 2-21
DECIMAL(p) data type 2-21
icon 9
Information Schema views 1-80
isolation level 1-86
level 15
public synonyms 1-67, 1-70
ANY operator 2-70
Arabic locales 2-12

archecker utility 3-24
Archiving
  setting DBREMOTECMD 3-45
Arithmetic
  DATE operands 2-16, 2-54
  DATETIME operands 2-52
  integer operands 2-25, 2-26, 2-41, 2-43
  INTERVAL operands 2-27, 2-53
  operators 2-70
  string operands 2-13
  time operands 2-51
AS keyword 2-66, 2-67
ASCII code set 1-41
AT keyword 2-29
Attached indexes 1-45, 3-31, 3-90
Audit analysis officer (AAO) 3-73
Authentication information file 3-67
Authorization identifier 1-76, 1-86

B

Backslash (\ ) symbol 3-35
BETWEEN operator 2-70
bin subdirectory 3-10
Binding style 1-85
BLOB data type
  casting not available 2-10
  coltype code 1-31
  description 2-9
  inserting data 2-10
  syscolattribs data 1-26
Blobspace
  defined 2-49
  memory cache for staging 3-68
  names (character set) 3-54
  sysblobs data 1-22
Boldface type 7
BOOLEAN data type
  coltype code 1-31
  description 2-10
Boolean expression
  with BOOLEAN data type 2-11
  with BYTE data type 2-11
  with TEXT data type 2-44
Borland C compiler 3-63
Bourne shell 3-9
Bracket ([ ]) symbols 2-12, 2-45
B-tree access method 1-21, 1-47, 3-53
Buffer
  BYTE or TEXT storage (DBBLOBBUF) 3-28
  fetch buffer (FET_BUFFER_SIZE) 3-55
  floating-point display (DBFLTMASK) 3-36
  network buffer (IFX_NETBUF_SIZE) 3-60
  private network buffer pool 3-60
Built-in access method 1-18, 1-21
Built-in aggregates 1-17, 2-30, 2-33, 2-42
Built-in casts 1-23, 2-63
Built-in data types
  casts 2-63, 2-69
  listed 2-48
  syscolumns.coltype code 1-29
  sysdistrib.type code 1-38
  sysxtdtypes data 1-79
BY keyword 2-11, 2-44
BYTE data type
  casting to BLOB 2-12
  description 2-11
  increasing buffer size 3-28
  inserting values 2-11
  restrictions
    in Boolean expression 2-11
    systables.npused 1-71
    with GROUP BY 2-11
    with LIKE or MATCHES 2-11
    with ORDER BY 2-11
  selecting from BYTE columns 2-12
  setting buffer size 3-28
  sysblobs data 1-22
  syscolumns data 1-33
  sysfragments data 1-43
  sysopclstr data 1-54
B+ tree index 1-45

C

C compiler
  default name 3-63
  INFORMIXC setting 3-63
  thread package 3-94
C shell 3-9
  .cshrc file 3-9
  .login file 3-9
call_type table in stores_demo database A-7
call_type table in superstores_demo database B-8
CARDINALITY( ) function 2-30, 2-33, 2-42
Cascading delete 1-63
Cast 2-62 to 2-69
  built-in 1-23, 2-63 to 2-67
  distinct data type 2-68
  explicit 1-23, 2-66, 2-67
  from BYTE to BLOB 2-12
  from TEXT to CLOB 2-45
  implicit 1-23, 2-66, 2-67
  rules of precedence 2-67
  syscasts data 1-23
CAST AS keywords 2-66
Cast (::) operator 2-66, 2-70
CHAR data type
  built-in casts 2-64
  collation 2-12, 2-13, 2-48
  conversion to NCHAR 2-34, 3-40, 3-41
  description 2-12
  multibyte values 2-14
  nonprintable characters 2-13
  storing numeric values 2-13
Character data types
  Boolean comparisons 2-47
  casting between 2-63
  data strings 2-8
  listed 2-48
  syscolumns data 1-32
CHARACTER data type. *See* CHAR data type.
Character string
  CHAR data type 2-12
  CHARACTER VARYING data type 2-14
  CLOB data type 2-14

DATETIME literals 2-19, 2-55, 3-51
INTERVAL literals 2-28
LVARCHAR data type 2-31
NCHAR data type 2-34
NVARCHAR data type 2-34
TEXT data type 2-44
VARCHAR data type 2-46
with DELIMIDENT set 3-54
CHARACTER VARYING data type
description 2-14
length (syscolumns) 1-32
See also VARCHAR data type.
Character-based applications 3-72, 3-92
Check constraint
creation-time value 3-31, 3-34
syschecks data 1-24
syscheckudrdep data 1-25
syscoldepend data 1-28
sysconstraints data 1-35
chkenv utility 3-8
error message 3-12
syntax 3-11
Chunk 2-49, 3-46
Client/server
Datablade API 2-51
default database 3-68
INFORMIXSQLHOSTS environment variable 3-70
shared memory communication segments 3-69
stacksize for client session 3-71
CLIENT_LOCALE environment variable 3-33
CLOB data type
casting not available 2-15
code-set conversion 2-15
collation 2-15
coltype code 1-31
description 2-14
inserting data 2-15
multibyte characters 2-15
syscolattribs data 1-26
CLOSE statement 3-83
Clustering 1-19, 1-45, 1-48
Code set
ASCII 1-41
collation order 2-47
conversion 3-96
East Asian 2-14, 2-47, 3-52
EBCDIC 1-41, 1-86
ISO 8859-1 1-39
Code set, ISO 8859-1 5
Code, sample, conventions for 11
Collation
CHAR data type 2-12, 2-13
CLOB data type 2-15
GL_COLLATE table 1-71
NCHAR data type 2-34
server_attribute data 1-86
TEXT data type 2-45
VARCHAR data type 2-47
Collection data type
casting matrix 2-69
description 2-59
LIST 2-29
MULTISET 2-32
SET 2-41
sysattrtypes data 1-21
sysxtddesc data 1-78
sysxtdtypes data 1-78, 1-79
Colon
cast (::) operator 2-66
DATETIME delimiter 2-18
INTERVAL delimiter 2-28
pathname separator 3-79
Color and intensity screen attributes 3-72
Column
changing data type 2-6, 2-62
constraints (sysconstraints) 1-34
default values (sysdefaults) 1-35
hashed 1-44
in sales_demo database B-3 to B-6
in stores_demo database A-2 to A-8
in superstores_demo database B-8 to B-22
inserting BLOB data 2-10
range of values 1-33
referential constraints (sysreferences) 1-63
syscolumns data 1-29
Column-level privileges
systabauth data 1-10
systabauth table 1-69
Combine function 1-17
Comment icons 8
Comment indicator 3-8
Comment lines 3-8
Committed read 1-86
Communications support module 3-63, 3-68
Commutator function 1-60
Compiling
ESQL/C programs 3-25
INFORMIXC setting 3-63
JAVA_COMPILER setting 3-78
multithreaded ESQL/C applications 3-94
Complex data type 2-58 to 2-61
collection types 2-59
ROW types 2-60
sysattrtypes data 1-21
Compliance
ANSI/ISO standard for SQL 1-80, 3-27
icons 9
sql_languages.conformance 1-84
with industry standards 15
XPG4 standard 1-83
X/Open CAE standards 1-80
Composite index 1-47
Concatenation (| |) operator 2-70
concsm.cfg file 3-63
Configuration file
for communications support module 3-64, 3-67
for connectivity 3-62, 3-68, 3-70
for database servers 3-54, 3-80
for High-Performance Loader 3-87
for MaxConnect 3-62
for ON-Bar utility 3-24
for onxfer utility 3-95
for terminal I/O 3-72, 3-93
.cshrc file 3-9
.informix 3-8, 3-12, 3-55, 3-57
.login file 3-9
.profile file 3-9
Configuration parameter
COSERVER 3-81
DBSPACETEMP 3-47
DEF_TABLE_LOCKMODE 3-57
DIRECTIVES 3-58

END 3-81
MITRACE_OFF 1-72, 1-73
NODE 3-81
OPCACHEMAX 3-68
OPTCOMPIND 3-82
OPT_GOAL 3-84
STACKSIZE 3-71
STMT_CACHE 3-92
USEOSTIME 2-20
CONNECT DEFAULT statement 3-69
Connect privilege 1-12, 1-76
CONNECT statement 3-42, 3-65, 3-69
Connection
authentication 3-67
coserver 3-69
INFORMIXCONRETRY environment variable 3-64
INFORMIXCONTIME environment variable 3-65
INFORMIXSERVER environment variable 3-68
Connectivity information 3-61, 3-70
Constraint
check
creation-time value 3-34
loading performance B-3
syschecks data 1-24
syscheckudrdep data 1-25
syscoldepend data 1-28
column
sysconstraints data 1-34
not null
collection data types 2-30, 2-33, 2-42, 2-59
syscoldepend data 1-28
syscolumns data 1-30
sysconstraints data 1-34
object mode 1-52
primary key
sysconstraints data 1-34
sysreferences data 1-63
unique SERIAL values 2-39
unique SERIAL8 values 2-40
referential
stores_demo data A-10
superstores_demo data B-25
sysconstraints data 1-34
sysreferences data 1-63
table
sysconstraints data 1-34
unique
sysconstraints data 1-34
violations 1-77
Constructor 2-42, 2-60
Contact information 15
Converting data types
CHAR and NCHAR 3-40
DATE and DATETIME 2-65
INTEGER and DATE 2-65
number and string 2-64
number to number 2-64
retyping a column 2-62
VARCHAR and NVARCHAR 3-40
Coserver
sysexternal data 1-42
sysviolations data 1-77
COSERVER configuration parameter 3-69, 3-81
CPFIRST environment variable 3-25
CREATE ACCESS METHOD statement 1-18
CREATE CAST statement 1-23, 2-66
CREATE DATABASE statement 3-42
CREATE DISTINCT TYPE statement 1-79, 2-24, B-22
CREATE EXTERNAL TABLE statement 1-40, 1-41
CREATE IMPLICIT CAST statement B-22
CREATE INDEX statement 1-46, 1-48, 1-51, 1-64, 1-71, 3-53
CREATE OPAQUE TYPE statement 2-35
CREATE OPERATOR CLASS statement 1-53
CREATE OPTICAL CLUSTER statement 1-55
CREATE PROCEDURE statement 1-65, 3-80
CREATE ROLE statement 1-65
CREATE ROUTINE FROM statement 1-65, 3-80
CREATE ROW TYPE statement 1-30, 2-36
CREATE SCHEMA statement 1-6
CREATE SYNONYM statement 1-66
CREATE TABLE statement
assigning data types 2-6
default lock mode 3-57
default privileges 3-80
SET constructor 2-42
typed tables 2-37
CREATE TEMP TABLE statement 3-46
CREATE TRIGGER statement 1-75
CREATE VIEW statement 1-9, 1-76
Credential 3-67
Currency symbol 2-32, 3-39
Current date 1-36, 3-29
CURRENT keyword 2-52
customer table in sales_demo database B-3
customer table in stores_demo database A-2
customer table in superstores_demo database B-10, B-11, B-12
cust_calls table in stores_demo database A-6
cust_calls table in superstores_demo database B-9
C++ map file 3-66

D

Data compression 3-74
Data corruption 1-12, 1-26
Data dependencies
syscheckudrdep data 1-25
syscoldepend data 1-28
sysdepend data 1-37
sysnewdepend data 1-51
Data dictionary 1-5
Data distributions 1-13, 1-37, 3-52
Data encryption 3-76
Data pages 1-26, 1-47, 1-70

Data type
- BLOB 2-9
- BOOLEAN 2-10
- BYTE 2-11
- CHAR 2-12
- CHARACTER 2-14
- CHARACTER VARYING 2-14
- CLOB 2-14
- DATE 2-16
- DATETIME 2-17
- DEC 2-21
- DECIMAL 2-21
- DISTINCT 2-23
- DOUBLE PRECISION 2-24
- FLOAT 2-24
- INT 2-25
- INT8 2-25
- INTEGER 2-26
- INTERVAL 2-26
- LIST 2-29
- LVARCHAR 2-31
- MONEY 2-31
- MULTISET 2-32
- NCHAR 2-34
- NUMERIC 2-34
- NVARCHAR 2-34
- OPAQUE 2-34
- REAL 2-35
- ROW 2-35, 2-37
- SERIAL 2-39
- SERIAL8 2-40
- SET 2-41
- SMALLFLOAT 2-43
- SMALLINT 2-43
- TEXT 2-44
- VARCHAR 2-46

Data types
- approximate 1-83
- casting 2-62 to 2-69
- classified by category 2-5
- collection 2-59
- complex 2-58
- conversion 2-62
- distinct 2-23, 2-61
- exact numeric 1-83
- extended 2-57
- fixed point 2-21
- floating-point 2-21, 2-25, 2-43
- inheritance 2-36
- internal 2-9
- named ROW 2-35
- opaque 2-34, 2-61
- sequential integer 2-40
- simple large object 2-49
- smart large object 2-50
- summary list 2-6
- unique numeric value 2-40
- unnamed ROW 2-37

Data warehousing B-1

Database
- data types 2-5
- dimensional B-2
- identifiers 3-53
- joins in stores_demo A-9
- object-relational B-1
- objects, sysobjectstate data 1-52
- privileges 1-76
- sales_demo B-1
- stores_demo A-1
- superstores_demo B-3, B-6
- syscrd 1-6
- sysmaster 1-6
- sysutils 1-6
- sysuuid 1-6

Database identifiers 3-54

Database server
- attributes in Information Schema view 1-85
- codeset 1-86
- coserver name 3-69
- default connection 3-68
- default isolation level 1-86
- optimizing queries 3-84
- pathname for 3-42
- remote 3-56
- role separation 3-73
- server name 1-36, 3-43

DATABASE statement 3-42

Database system administrator (DBSA) 1-6

Database system security officer (DBSSO) 3-73

DataBlade module
- Client and Server API 2-51
- data types (sysbuiltintypes) 1-6
- trace messages (systracemsgs) 1-72, 1-73
- user messages (syserrors) 1-39

DATE data type
- abbreviated year values 2-16, 3-28
- casting to integer 2-65
- converting to DATETIME 2-65
- description 2-16
- display format 3-32
- in expressions 2-51, 2-54
- international date formats 2-16
- source data 2-55

DATETIME data type
- abbreviated year values 3-28
- converting to DATE 2-65
- description 2-17
- display format 3-48
- EXTEND function 2-54
- extending precision 2-52
- field qualifiers 2-17
- in expressions 2-51 to 2-56
- international formats 2-20
- length (syscolumns) 1-33
- literal values 2-19
- precision and size 2-17
- source data 2-55
- two-digit year values and DBDATE variable 2-19
- year to fraction example 2-19

DATE( ) function 2-55, 3-34

DAY keyword
- DATETIME qualifier 2-17
- INTERVAL qualifier 2-27
- UNITS operator 2-16, 2-56

DBA privilege 1-40, 1-72, 1-73, 1-76

DBA routines 1-61

DB-Access utility 1-12, 1-81, 3-7, 3-36, 3-42, 3-48, 3-68

DBACCNOIGN environment variable 3-26

DBANSIWARN environment variable 3-27

DBBLOBBUF environment variable 3-28

DBCENTURY environment variable
- description 3-28
- effect on functionality of DBDATE 3-33
- expanding abbreviated years 2-16, 2-19, 3-29

DBDATE environment
variable 2-16, 3-32
DBDELIMITER environment
variable 3-35
DBEDIT environment variable 3-35
dbexport utility 3-35
DBFLTMASK environment
variable 3-36
DBLANG environment
variable 3-37
dbload utility 2-10, 2-11, 2-44, 3-35
DBMONEY environment
variable 2-32, 3-38
DBNLS environment variable 3-40
DBONPLOAD environment
variable 3-41
DBPATH environment
variable 3-42
DBPRINT environment
variable 3-44
DBREMOTECMD environment
variable 3-45
Dbserver group 3-69
DBSERVERNAME configuration
parameter 3-69
dbservername.cmd batch file 3-17
dbslice 1-44, 1-45
dbspace
for BYTE or TEXT values 1-22
for system catalog 1-6
for table fragments 1-42
for temporary tables 3-46
name 3-54
DBSPACE keyword 1-44
DBSPACETEMP configuration
parameter 3-46
DBSPACETEMP environment
variable 3-46
DBTEMP environment
variable 3-48
DBTIME environment
variable 2-20, 3-48
DBUPSPACE environment
variable 3-52
DCE-GSS communications support
module (CSM) 3-67
DEC data type. *See* DECIMAL data
type.
DECIMAL data type
built-in casts 2-64
description 2-21
disk storage 2-22
display format 3-36, 3-38
fixed point 2-21
floating point 2-21
length (syscolumns) 1-32
Decimal digits, display of 3-36
Decimal point
DBFLTMASK setting 3-36
DBMONEY setting 3-39
DECIMAL radix 2-22
Decimal separator 3-39
DECLARE statement 3-83
Default
C compiler 3-63
century 3-29, 3-50
CHAR length 2-12
character set for SQL
identifiers 3-53
compilation order 3-25
configuration file 3-80
connection 3-69
data type 2-38
database server 3-43, 3-68
DATE display format 2-16
DATE separator 3-33
DATETIME display format 2-20
DECIMAL precision 2-21
detached indexes 3-53
detail level 3-75
disk space for sorting 3-52
fetch buffer size 3-55
heap size 3-78
isolation level 1-86
join method 3-81
level of parallelism 3-86
lock mode 3-57
message directory 3-37
MONEY scale 2-32
operator class 1-18, 1-53
printing program 3-44
query optimizer goal 3-83
sysdefaults.default 1-35
table privileges 3-80
temporary dbspace 3-47
termcap file 3-93
text editor 3-35
Default locale 5
DEFAULT_ATTACH environment
variable 3-53
DEFINE statement of SPL 2-39,
2-40
DEF_TABLE_LOCKMODE
configuration parameter 3-57
Delete privilege 1-43, 1-68, 3-80
DELETE statement 1-13, 1-77
Delete trigger 1-75
DELIMIDENT environment
variable 3-53
DELIMITED files 1-40, 1-41
Delimited identifiers 3-53
Delimiter
for DATETIME values 2-18
for fields 1-41, 3-35
for identifiers 3-53
for INTERVAL values 2-28
Demonstration database
structure of tables B-7
tables A-2 to A-8
Dependencies, software 4
Descending index 1-47
DESCRIBE statement 3-61
Describe-for-updates 3-61
Detached index 3-53
Deutsche mark (DM) currency
symbol 3-39
Diagnostics table 1-77
DIRECTIVES configuration
parameter 3-58
Directives for query
optimization 3-58, 3-81, 3-84
Disabled object 1-77
Disk space
for data distributions 3-52
for temporary data 3-47
Distinct data type
casts 2-68
description 2-23
sysxtddesc data 1-78
sysxtdtypes data 1-79, 1-80, 2-24
Distributed Computing
Environment (DCE) 3-94
Distributed query 3-56
Documentation notes 13
Documentation notes, program
item 14

Documentation, types of 12
documentation notes 13
machine notes 13
release notes 13
Dollar sign 2-32, 3-38
Double data type of C 2-24
DOUBLE PRECISION data type. *See* FLOAT data type.
Double-precision floating-point number 2-24
DROP CAST statement B-22
DROP DATABASE statement 3-42
DROP FUNCTION statement 1-61
DROP INDEX statement 1-71
DROP OPTICAL CLUSTER statement 1-55
DROP PROCEDURE statement 1-61
DROP ROUTINE statement 1-61
DROP ROW TYPE statement 2-36
DROP TYPE statement 2-24, 2-35
DROP VIEW statement 1-81

E

EBCDIC collation 1-41, 1-86
Editor, DBEDIT setting 3-35
EMACS text editor 3-36
Empty set 2-60
Encryption 3-76
END configuration parameter 3-81
Enterprise Replication 1-6
env utility 3-11
ENVIGNORE environment variable
description 3-8, 3-54
relation to chkenv utility 3-12
Environment configuration file
debugging with chkenv 3-12
setting environment variables in UNIX 3-6, 3-8
Environment registry key 3-13
Environment variable
AC_CONFIG 3-24
AFDEBUG 3-24
C8BITLEVEL 3-19
CLIENT_LOCALE 3-19, 3-33
CPFIRST 3-25
DBACCNOIGN 3-26
DBANSIWARN 3-27
DBBLOBBUF 3-28
DBCENTURY 2-16, 3-28
DBDATE 2-16, 3-32
DBDELIMITER 3-35
DBEDIT 3-35
DBFLTMASK 3-36
DBLANG 3-37
DBMONEY 2-32, 3-38
DBNLS 3-40
DBONPLOAD 3-41
DBPATH 3-42
DBPRINT 3-44
DBREMOTECMD 3-45
DBSPACETEMP 3-46
DBTEMP 3-48
DBTIME 2-20, 3-48
DBUPSPACE 3-52
DB_LOCALE 3-20
DEFAULT_ATTACH 3-53
DELIMIDENT 3-53
ENVIGNORE 3-54
ESQLMF 3-20
FET_BUF_SIZE 3-55
GLS8BITSYS 3-20
GL_DATE 2-16, 3-32
GL_DATETIME 2-20, 3-32
IFMX_SMLTBL_BROADCAST_SIZE 3-56
IFX_DEF_TABLE_LOCKMODE 3-57
IFX_DIRECTIVES 3-58
IFX_LONGID 3-59
IFX_NETBUF_PVTPOOL_SIZE 3-60
IFX_NETBUF_SIZE 3-60
IFX_UPDDESC 3-61
IMCADMIN 3-61
IMCCONFIG 3-62
IMCSERVER 3-62
INFORMIXC 3-63
INFORMIXCONCSMCFG 3-63
INFORMIXCONRETRY 3-64
INFORMIXCONTIME 3-65
INFORMIXCPPMAP 3-66
INFORMIXDIR 3-67
INFORMIXKEYTAB 3-67
INFORMIXOPCACHE 3-68
INFORMIXSERVER 3-68
INFORMIXSHMBASE 3-69
INFORMIXSQLHOSTS 3-70
INFORMIXSTACKSIZE 3-71
INFORMIXTERM 3-72
INF_ROLE_SEP 3-73
INTERACTIVE_DESKTOP_OFF 3-74
ISM_COMPRESSION 3-74
ISM_DEBUG_FILE 3-75
ISM_DEBUG_LEVEL 3-75
ISM_ENCRYPTION 3-76
ISM_MAXLOGSIZE 3-76
ISM_MAXLOGVERS 3-77
JAR_TEMP_PATH 3-77
JAVA_COMPILER 3-78
JVM_MAX_HEAP_SIZE 3-78
LD_LIBRARY_PATH 3-79
LIBPATH 3-79
NODEFDAC 3-80
ONCONFIG 3-80
OPTCOMPIND 3-81
OPTMSG 3-82
OPTOFC 3-83
OPT_GOAL 3-83
PATH 3-84
PDQPRIORITY 3-85
PLCONFIG 3-87
PLOAD_LO_PATH 3-87
PLOAD_SHMBASE 3-88
PSORT_DBTEMP 3-89
PSORT_NPROCS 3-89
SERVER_LOCALE 3-22
SHLIB_PATH 3-91
STMT_CACHE 3-92
TERM 3-92
TERMCAP 3-93
TERMINFO 3-94
THREADLIB 3-94
XFER_CONFIG 3-95
Environment variables 7
command-line utilities 3-15
displaying current settings 3-11, 3-16
how to set
in Bourne shell 3-9
in C shell 3-9
in Korn shell 3-9
how to set in Bourne shell 3-9

how to set in Korn shell 3-9
listed alphabetically 3-18
listed by topic 3-95
manipulating in Windows environments 3-13
modifying settings 3-10
overriding a setting 3-8, 3-54
rules of precedence in UNIX 3-12
rules of precedence in Windows 3-18
scope of reference 3-15
setting
at the command line 3-6
for native Windows applications 3-13
in a configuration file 3-6
in a login file 3-6
in a shell file 3-9
in Windows environments 3-7
with command-line utilities 3-15
with the Registry Editor 3-13
with the System applet 3-15
setting in autoexec.bat 3-16
standard UNIX system 3-5
types of 3-5
unsetting 3-10, 3-16, 3-54
view current setting 3-11
where to set 3-9
en_us.8859-1 locale 5
Equality (=) operator 2-15
Era-based dates 3-52
Error message files 3-37
esql command 3-25, 3-63
ESQL/C
DATETIME routines 3-49
esqlc command 3-25
long identifiers 3-59
message chaining 3-82
multithreaded applications 3-94
program compilation order 3-25
Exact numeric data types 1-83
Executable programs 3-84
Execute privilege 1-56, 3-80
Explicit cast 1-23, 2-66
Explicit pathnames 3-16, 3-44
Explicit temporary tables 3-46
Exponent 2-22
export utility 3-9
Expression-based fragmentation 1-44, 3-31, 3-34
EXTEND function 2-54
Extended data types 1-79, 2-57, B-22
Extended Parallel Server (XPS) 1-14, 3-18, B-1
Extensible Markup Language (XML) 2-14
Extension checking (DBANSIWARN) 3-27
Extension, to SQL, symbol for 9
Extent, changing size 1-12
External database 1-66
External database server 1-66
External routine 1-59
External sequence 1-67
External table
sysextcols data 1-40
sysextdfiles data 1-41
sysexternal data 1-41
syssyntable data 1-67
systables data 1-70
External view 1-67
extspace 1-18

F

Fact table B-3
FALSE setting
Boolean value 2-10
CPFIRST 3-25
ISM_COMPRESSION 3-75
Farsi locales 2-12
Feature icons 8
Features in 9.3 6
Fetch buffer 3-55
Fetch buffer size 3-55
FETCH statement 3-83
FET_BUF_SIZE environment variable 3-55
Field delimiter
DBDELIMITER 3-35
Field of a ROW data type 2-60
Field qualifier
DATETIME values 2-17
EXTEND function 2-54
INTERVAL values 2-26
File
environment configuration files 3-12
installation directory 3-67
permission settings 3-8
shell 3-9
temporary 3-46, 3-48, 3-89
temporary for SE 3-48
termcap, terminfo 3-72, 3-93, 3-94
File extensions
.a 3-59
.cfg 3-63
.cmd 3-17
.ec 3-25
.ecp 3-25
.iem 3-37
.jar 3-77
.rc 3-8, 3-12, 3-54, 3-57
.so 3-59
.sql 1-81, 3-42, 3-54, B-2, B-6
.std 3-24, 3-81, 3-92
.xps 3-81
FILETOBLOB function 2-10
FILETOCLOB function 2-15
Filtering mode 1-52, 1-77
Finalization function 1-17
finderr utility 14
FIXED column format 1-40, 1-41
Fixed point decimal 2-21, 2-31, 3-39
Fixed-length UDT 1-79
FLOAT data type
built-in casts 2-64
description 2-24
display format 3-36, 3-38
Floating-point decimal 2-21, 2-24, 2-43, 3-36
Foreign key A-10, B-3
Formatting
DATE values with DBDATE 3-32
DATE values with GL_DATE 3-52
DATETIME values with DBTIME 3-48
DATETIME values with GL_DATETIME 3-52
DECIMAL(p) values with DBFLTMASK 3-36
FLOAT values with DBFLTMASK 3-36

MONEY values with DBMONEY 3-38
SMALLFLOAT values with DBFLTMASK 3-36
Formatting mask
with DBDATE 3-32
with DBFLTMASK 3-36
with DBMONEY 3-38
with DBTIME 3-49
with GL_DATE 3-48
with GL_DATETIME 3-48
FRACTION keyword
DATETIME qualifier 2-18
INTERVAL qualifier 2-27
FRAGMENT BY clause 3-46
Fragmentation
distribution strategy 1-44
expression 1-44, 3-31, 3-34
list 1-44
PDQPRIORITY environment variable 3-86
PSORT_NPROCS environment variable 3-90
round robin 1-44
setting priority levels for PDQ 3-85
sysfragauth data 1-42
sysfragments data 1-43
FROM keyword 1-12, 1-24
Function
for BLOB columns 2-10
for CLOB columns 2-15
for MULTISET columns 2-33
support for complex types 2-59
Function keys 3-72
Functional index 1-47, 2-60
fwritable gcc option 3-63

G

gcc compiler 3-63
Generalized-key index
sysindexes data 1-46
sysnewdepend data 1-51
sysrepository data 1-64
Generic B-trees 1-47
geography table in sales_demo database B-4
GET DIAGNOSTICS statement 1-39
Global Language Support (GLS) 5, 3-40
Global network buffer pool 3-60
Globally detached index 1-44
GLS environment variables 3-13
GL_COLLATE table 1-71
GL_CTYPE table 1-71
GL_DATE environment variable 2-16, 3-32, 3-33
GL_DATETIME environment variable 2-20, 3-32
GNU C compiler 3-63
GRANT statement 1-65
Graphic characters 3-72
GROUP BY clause 2-11, 2-44, 3-47
Group informix 3-37

H

Hash join 3-81
Hashed columns 1-44
Hashing parameters 1-67
Heap size 3-78
Hebrew locales 2-12
Help 12
Hexadecimal digits 3-35
HIGH INTEG keywords
ALTER TABLE statement 2-50
CREATE TABLE statement 2-50
HIGH keyword
PDQPRIORITY 3-85
UPDATE STATISTICS 1-13, 1-38
High-Performance Loader 3-41, 3-87
Histogram 1-38
HKEY_LOCAL_MACHINE window 3-13
Host variable 2-10, 2-11, 2-44, 2-60
HOUR keyword
DATETIME qualifier 2-17
INTERVAL qualifier 2-27
HP-UX operating system 3-91
HTML (Hypertext Markup Language) 2-14
Hybrid fragmentation strategy 1-44
Hyphen
DATETIME delimiter 2-18
INTERVAL delimiter 2-28

I

Icons
compliance 9
feature 8
Important 8
platform 8
product 8
Tip 8
Warning 8
IDS (Informix Dynamic Server) 1-14
IFMX_SMLTBL_BROADCAST_SIZE environment variable 3-56
IFX_DEF_TABLE_LOCKMODE environment variable 3-56
IFX_DIRECTIVES environment variable 3-58
IFX_LONGID environment variable 3-59
IFX_NETBUF_PVTPOOL_SIZE environment variable 3-60
IFX_NETBUF_SIZE environment variable 3-60
IFX_UPDDESC environment variable 3-61
imcadmin administrative tool 3-61
IMCADMIN environment variable 3-61
IMCCONFIG environment variable 3-62
IMCSERVER environment variable 3-62
IMPEX data type 2-67
IMPEXBIN data type 2-67
Implicit cast 1-23, 2-66
Implicit connection 3-69
Implicit temporary tables 3-46
Important paragraphs, icon for 8
IN keyword 1-44, 2-11, 2-33, 2-38, 2-42, 2-44, 2-70
Index
attached 1-45, 3-31, 3-53, 3-90
B-tree 1-47, 3-53

clustered 1-46, 1-48
composite 1-46, 1-47
default values for attached 3-90
descending 1-47
detached 3-53
fragmented 1-43
functional 1-47, 2-60
generalized-key 1-46, 1-51, 1-64
globally detached 1-44
nonfragmented 3-53
of data types 2-6
of environment variables 3-95
of system catalog 1-14
R-tree 3-53
sysindexes data 1-46
sysindices data 1-48
sysobjstate data 1-52
threads for sorting 3-90
unique 1-35, 1-46, 2-39, 2-40
Index privilege 1-68
Indexkey structure 1-49
Indirect typing 2-39, 2-40
Industry standards
See Compliance.
Industry standards, compliance with 15
Information Schema views
accessing 1-81
columns 1-83
description 1-80
generating 1-81
server_info 1-85
sql_languages 1-84
tables 1-82
Informational messages 1-39
Informix Dynamic Server (IDS) 1-14
Informix ESQL/C 3-25, 3-33, 3-49, 3-59, 3-82
Informix Extended Parallel Server (XPS) 1-14, 3-18
Informix extension checking (DBANSIWARN) 3-27
informix owner name 1-12, 1-23, 1-38, 1-48, 1-69, 3-37, 3-73
Informix Storage Manager (ISM) 3-74, 3-77
Informix subkey 3-13
INFORMIXC environment variable 3-63
INFORMIXCONCSMCFG environment variable 3-63
INFORMIXCONRETRY environment variable 3-64
INFORMIXCONTIME environment variable 3-65
INFORMIXCPPMAP environment variable 3-66
INFORMIXDIR environment variable 3-67
INFORMIXDIR/bin directory 6
INFORMIXKEYTAB environment variable 3-67
INFORMIXOPCACHE environment variable 3-68
INFORMIXSERVER environment variable 3-68
INFORMIXSHMBASE environment variable 3-69
INFORMIXSTACKSIZE environment variable 3-71
INFORMIXTERM environment variable 3-72
informix.rc file 3-8, 3-12, 3-57
INF_ROLE_SEP environment variable 3-73
Inheritance hierarchy 1-50, 2-37
Initialization function 1-17, 1-65
Input support function 2-31
Insert privilege 1-43, 1-68, 3-80
INSERT statement 1-72, 1-77, 2-19, 2-60, 3-26, 3-34
Insert trigger 1-75
Installation directory 3-67
INT data type. *See* INTEGER data type.
INT8 data type
built-in casts 2-64
description 2-25
using with SERIAL8 2-41
INTEG keyword 2-50
INTEGER data type
built-in casts 2-64
description 2-26
length (syscolumns) 1-32
Intensity attributes 3-72
INTERACTIVE_DESKTOP_OFF environment variable 3-74
Internationalized trace messages 1-73
Interprocess communications (IPC) 3-69
INTERVAL data type
description 2-26
field delimiters 2-28
in expressions 2-51, 2-56 to 2-57
length (syscolumns) 1-33
ipcshm protocol 3-69
IS NULL operator 2-11, 2-44
ISM_COMPRESSION environment variable 3-74
ISM_DEBUG_FILE environment variable 3-75
ISM_DEBUG_LEVEL environment variable 3-75
ISM_ENCRYPTION environment variable 3-76
ISM_MAXLOGSIZE environment variable 3-76
ISM_MAXLOGVERS environment variable 3-77
ISO 8859-1 code set 5, 1-86
Isolation level 1-86, 3-81
items table in stores_demo database A-4
items table in superstores_demo database B-13
Iterator function 1-17, 1-60

J

Japanese eras 3-52
Jar management procedures 3-77
JAR_TEMP_PATH environment variable 3-77
Java virtual machine (JVM) 3-24, 3-77, 3-78
JAVA_COMPILER environment variable 3-78
JIT compiler 3-78
Join columns A-8, B-25
Join methods 3-81
Join operations 1-13, 3-47

JVM_MAX_HEAP_SIZE
environment variable 3-78

K

KEEP ACCESS TIME keywords
ALTER TABLE statement 2-50
CREATE TABLE statement 2-50
Key
foreign A-10, B-3
generalized 1-51, 1-64
primary 1-34, 1-63, 1-77, A-10, B-8
Key scan 1-19
Key tables 3-67
Keyboard I/O
INFORMIXTERM setting 3-72
TERM setting 3-92
TERMCAP setting 3-93
TERMINFO setting 3-94
keytab file 3-67
Korn shell 3-9

L

Language
C 1-65, 3-25, 3-63
CLIENT_LOCALE setting 3-13, 3-33
C++ 3-66
DBLANG setting 3-37
Extensible Markup Language (XML) 2-14
Hypertext Markup Language (HTML) 2-14
Informix ESQL/C 2-51, 2-60, 3-94
Java 3-24, 3-77, 3-78
sql_languages information schema view 1-84
Stored Procedure Language (SPL) 2-60, 3-31, 3-34
syslangauth data 1-50
sysroutinelangs data 1-65
See also Locale.
Large-object data type
description 2-48
listed 2-48
LD_LIBRARY_PATH environment variable 3-79
Leaf pages 1-45
libos.a library 3-59
LIBPATH environment variable 3-79
LIKE keyword of SPL 2-39, 2-40
LIKE operator 2-11, 2-44, 2-70
Linearized code 1-74
List
of data types 2-6
of environment variables 3-18
of environment variables, by topic 3-95
of system catalog tables 1-14
LIST data type, description 2-29
LOAD statement 2-10, 2-11, 2-44, 3-35
Locale 5
collation order 1-71, 2-48
default 5
en_us.8859-1 5
multibyte 2-14
of trace messages 1-73
right-to-left 2-12
specifying 3-96, 3-101
LOCKMODE keyword 3-56
Lock-table overflow 3-57
LOCOPY function 2-10, 2-15
LOG keyword
ALTER TABLE statement 2-50
CREATE TABLE statement 2-50
Logging mode 1-26
Long identifiers
client version 3-59
IFX_LONGID setting 3-59
Information Schema views 1-82
LOTOFILE function 2-10, 2-15
LOW keyword
PDQPRIORITY 3-85
UPDATE STATISTICS 1-38
Lowercase mode codes 1-61
Lowercase privilege codes 1-10, 1-27, 1-42, 1-68, 1-78
LVARCHAR data type
casting opaque types 2-67
coltype code 1-31
description 2-31

M

Machine notes 13, 3-72
Magnetic storage media 1-22
manufact table in superstores_demo database B-17
Map file for C++ programs 3-66
MATCHES operator 2-11, 2-44, 2-48, 2-70
MaxConnect 3-61, 3-62
MEDIUM keyword 1-13, 1-38
Membership operator 2-70
Memory cache, for staging blobspace 3-68
Message file
specifying subdirectory with DBLANG 3-37
XBSA 3-75
Message file for error messages 14
Messages
chaining 3-82
error in syserrors 1-39
optimized transfers 3-82
reducing requests 3-83
trace message template 1-73
warning in syserrors 1-39
Microsoft C compiler 3-63
MINUTE keyword
DATETIME qualifier 2-18
INTERVAL qualifier 2-27
MITRACE_OFF configuration parameter 1-72, 1-73
mi_collection_card() function 2-30, 2-33, 2-42
mi_db_error_raise() function 1-39
mkdir utility 3-37
MODERATE INTEG keywords
ALTER TABLE statement 2-50
CREATE TABLE statement 2-50
Modifiers
CLASS 1-60
COSTFUNC 1-60
HANDLESNULLS 1-60
INTERNAL 1-60
ITERATOR 1-60
NEGATOR 1-60
NOT VARIANT 1-60
PARALLELIZABLE 1-60

SELCONST 1-60
STACK 1-60
VARIANT 1-60
MODIFY NEXT SIZE
keywords 1-12
MONEY data type
built-in casts 2-64
description 2-31
display format 3-38
international money formats 2-32
length (syscolumns) 1-32
MONTH keyword
DATETIME qualifier 2-17
INTERVAL qualifier 2-27
Multibyte characters
CHAR data type 2-14
CLOB data type 2-15
TEXT data type 2-45
VARCHAR data type 2-47
MULTISET data type
constructor 2-60
description 2-32

N

Named ROW data type
casting permitted 2-69
defining 2-35
description 2-35
equivalence 2-36
inheritance 1-50, 2-36
typed tables 2-37
See also ROW type.
NCHAR data type
collation order 2-34
conversion to CHAR 3-41
description 2-34
multibyte characters 2-34
Negator function 1-60
Nested dot notation 2-60
Nested-loop join 3-81
Network buffers 3-60
Network environment variable,
DBPATH 3-42
NFS directory 3-48
NO KEEP ACCESS TIME keywords
ALTER TABLE statement 2-50
CREATE TABLE statement 2-50
no setting of NODEFDAC 3-80
NODE configuration
parameter 3-81
NODEFDAC environment
variable 3-80
NOLOG keyword
ALTER TABLE statement 2-50
CREATE TABLE statement 2-50
NONE setting
ISM_ENCRYPTION 3-76
JAVA_COMPILER 3-78
Nonprintable characters
CHAR data type 2-13
TEXT data type 2-45
VARCHAR data type 2-47
Not null constraint
collection elements 2-30, 2-33,
2-42, 2-59
syscoldepend data 1-28
sysconstraints data 1-35
NOT NULL keywords 2-11, 2-30,
2-44
NOT operator 2-70
NULL value
allowed or not allowed 1-30
BYTE data type 2-11
TEXT data type 2-44
NULLvalue
allowed or not allowed 1-17
BOOLEAN literal 2-10
Numeric data types
casting between 2-64
casting to character types 2-64
listed 2-48
NUMERIC data type. *See*
DECIMAL data type.
NVARCHAR data type
collation order 2-34
conversion to VARCHAR 3-41
description 2-34
length (syscolumns) 1-32
multibyte characters 2-34

O

Object mode of database
objects 1-52
Object-relational schema B-1
ODBC driver 3-79, 3-91
OFF setting
IFX_DIRECTIVES 3-58
PDQPRIORITY 3-85
ON setting
IFX_DIRECTIVES 3-58
ON-Bar 3-75
ONCONFIG environment
variable 3-80
onconfig.std file 3-81, 3-92
onconfig.xps file 3-81
oninit command 3-57
Online help 12
Online manuals 12
onload utility 2-10, 2-11, 2-44
onpload utility 3-41, 3-88
Opaque data type
cast matrix 2-69
comparing 2-67
description 2-34
smart large objects 2-50
storage 2-31
sysxtddesc data 1-78
sysxtdtypes data 1-79
OPCACHEMAX configuration
parameter 3-68
OPEN statement 3-83
Operator class
sysams data 1-18
sysindices data 1-49
sysopclasses data 1-53
Operator precedence 2-70
OPTCOMPIND configuration
parameter 3-82
OPTCOMPIND environment
variable 3-81
Optical cluster
INFORMIXOPCACHE
setting 3-68
sysblobs.type 1-22
sysopclstr data 1-54
Optimizer
setting IFX_DIRECTIVES 3-58
setting OPTCOMPIND 3-82
setting OPTOFC 3-83
setting OPT_GOAL 3-83
sysdistrib data 1-38
OPTMSG environment
variable 3-82

OPTOFC environment
variable 3-83
OPT_GOAL configuration
parameter 3-84
OPT_GOAL environment
variable 3-83
OR operator 2-70
ORDER BY clause 2-11, 2-44, 3-47
orders table in superstores_demo
database B-14, B-15, B-16, B-18
Ordinal positions 2-29
Output support function 2-31
Overflow error 2-21
Owner routines 1-61, 3-80

P

PAGE lock mode 1-70, 3-57
Parallel database query. *See* PDQ.
Parallel distributed queries, setting
with PDQPRIORITY 3-85
Parallel sorting, setting with
PSORT_NPROCS 3-89
Partial characters 2-12
PATH environment variable 3-84
Pathname
for C compiler 3-63
for client or shared libraries 3-79
for concsm.cfg file 3-64
for connectivity information 3-70
for C++ map file 3-66
for database server 3-42
for dynamic-link libraries 3-79,
3-91
for environment-configuration
file 3-12
for executable programs 3-84
for installation 3-67
for keytab file 3-67
for message files 3-37
for parallel sorting 3-89
for remote shell 3-45
for smart-large-object
handles 3-87
for temporary .jar files 3-77
for termcap file 3-93
for terminfo directory 3-94
for XBSA messages 3-75
for xfer_config file 3-95
separator symbols 3-84
PDQ
OPTCOMPIND environment
variable 3-81
PDQPRIORITY environment
variable 3-85
PDQPRIORITY configuration
parameter 3-86
Percentage (%) symbol 3-49
Period
DATE delimiter 3-33
DATETIME delimiter 2-18
INTERVAL delimiter 2-28
Permissions 3-8, 3-37
Platform icons 8
PLCONFIG environment
variable 3-87
plconfig file 3-87
PLOAD_LO_PATH environment
variable 3-87
PLOAD_SHMBASE environment
variable 3-88
PostScript 2-14
Precedence rules
for casts 2-67
for lock mode 3-57
for native Windows
application 3-18
for SQL operators 2-70
for UNIX environment
variables 3-12
for Windows environment
variables 3-18
Precision
of currency values 2-31
of numbers 2-21, 2-24, 2-25, 2-26,
2-43, 2-64
of time values 2-17, 2-26, 2-52,
2-57
PREPARE statement 1-71
Prepared statement 1-71
Primary access method 1-19, 1-67
Primary key 1-35, 1-63, 1-77, 2-39,
2-40, A-2, B-8
Primary thread 3-71
printenv utility 3-11
Printing with DBPRINT 3-45
Private environment-configuration
file 3-12, 3-55
Private network buffer pool 3-60
Private synonym 1-70
Privilege
default table privileges 3-80
on columns (syscolauth
table) 1-27
on procedures and functions
(sysprocauth table) 1-56
on table fragments (sysfragauth
table) 1-42
on tables (systabauth table) 1-68
on the database (sysusers
table) 1-76
on UDTs and named row types
(sysxtdtypeauth) 1-78
Product icons 8
product table in sales_demo
database B-4
Program group
Documentation notes 14
Release notes 14
Protected routines 1-61
Pseudo-machine code (p-
code) 1-57
PSORT_DBTEMP environment
variable 3-89
PSORT_NPROCS environment
variable 3-89
Public synonym 1-66, 1-70
public user name 1-81, 3-80
Purpose functions 1-19

Q

Qualifier field
DATETIME 2-17
EXTEND 2-56
INTERVAL 2-26
UNITS 2-56
Query optimizer
description 1-13
directives 3-58
sysprocplan data 1-62
Quoted string
DATE and DATETIME
literals 2-55

DELIMIDENT setting 3-54
INTERVAL literals 2-28
invalid with BYTE 2-12
invalid with TEXT 2-45
LVARCHAR data type 2-31

R

Raw UNIX devices 3-46
Read committed 1-86
Read uncommitted 1-86
REAL data type. *See* SMALLFLOAT data type.
References privilege 1-27, 1-68
Referential constraint 1-35, 1-63, 1-77, A-10, B-25
regedt32.exe Registry Editor 3-13
region table in superstores_demo database B-18
Registry Editor 3-13
Reject file 1-42
Relational operators 2-13, 2-70
Release notes 13
Release notes, program item 14
Remote database server 3-56
Remote shell 3-45
Remote tape devices 3-45
Repeatable read 3-81
Replica identifier 1-44
Resource contention 3-86
Resource privilege 1-12, 1-76
REVOKE statement 1-68
Right-to-left locales 2-12
Role
INF_ROLE_SEP setting 3-73
sysroleauth data 1-65
sysusers data 1-76
Role separation 3-73
Round-robin fragmentation 1-44
Routine
DataBlade API routine 1-72
DATETIME formatting 3-48
owner 1-58
privileges 1-56
protected 1-61
Stored Procedure Language (SPL) 2-60
syserrors data 1-39
syslangauth data 1-50
sysprocauth data 1-56
sysprocbody data 1-57
sysprocedures data 1-58
sysprocplan data 1-62
sysroutinelangs data 1-65
systraceclasses data 1-72
systracemsgs data 1-73
See also User-defined routine.
Routine identifier
description 1-58
ROW lock mode 1-70, 3-57
ROW type 2-60
casting permitted 2-69
equivalence 2-36
fields 1-21, 2-60
inheritance 1-50, 2-36
inserting values 2-38
named 2-35
sysattrtypes data 1-21
sysxtddesc data 1-78
sysxtdtypes data 1-78, 1-79
unnamed 2-37
Rowids 1-19
RTNPARAMTYPES data type 1-59
R-tree index 3-53
Runtime
warnings (DBANSIWARN) 3-27

S

sales table in sales_demo database B-5
sales_demo database
customer table columns B-3
description B-3
geography table columns B-4
product table columns B-4
sales table columns B-5
time table columns B-6
sales_rep table in superstores_demo database B-19
sbspace
defined 2-14, 2-50
name 3-54
sysams data 1-18
syscolattribs data 1-26
systabamdata data 1-68
Scale of numbers 2-21, 3-36
Scan cost 1-19
Schema Tools 3-7
SECOND keyword
DATETIME qualifier 2-18
INTERVAL qualifier 2-27
Secondary access method 1-19, 1-34, 1-49, 1-53, 2-34
SELECT INTO TEMP statement 3-47
Select privilege 1-27, 1-68, 1-81, 3-80
SELECT statement 1-12, 1-37
Select trigger 1-75
Selectivity constant 1-60
Self-join 1-9
SENDRECV data type 2-67
Sequence
syssynonyms data 1-66
syssyntable data 1-66
systabauth data 1-68
systables data 1-69
Sequential integers
aggid code 1-17
am_id code 1-18
classid code 1-72
constrid code 1-34
extended_id code 1-79
id code 1-39
langid code 1-65
msgid code 1-73
opclassid code 1-53
planid code 1-62
procid code 1-58, 1-59
SERIAL data type 2-39
SERIAL8 data type 2-40
tabid code 1-8, 1-69
trigid code 1-75
udr_id code 1-25
SERIAL data type
description 2-39
inserting values 2-39
length (syscolumns) 1-32
resetting values 2-39
SERIAL8 data type
assigning a starting value 2-40
description 2-40
inserting values 2-41

length (syscolumns) 1-32
resetting values 2-40
using with INT8 2-41
Serializable transactions 1-86
SET data type, description 2-41
SET OPTIMIZATION statement 3-84
SET PDQPRIORITY statement 3-86
SET SESSION AUTHORIZATION statement 1-61
SET STMT_CACHE statement 3-92
set utility 3-16
setenv utility 3-10
Setnet32 utility 3-7
Setting environment variables
in UNIX 3-8
in Windows 3-13
SGML (Standard Graphic Markup Language) 2-14
Shared environment-configuration file 3-12
Shared libraries 3-59
Shared memory
INFORMIXSHMBASE 3-69
PLOAD_SHMBASE 3-88
Shell
remote 3-45
search path 3-84
setting environment variables in a file 3-9
specifying with DBREMOTECMD 3-45
SHLIB_PATH environment variable 3-91
Simple large objects
description 2-49
length (syscolumns) 1-33
location (sysblobs) 1-22
Single-precision floating-point number 2-35, 2-43
Small Table Broadcast 3-56
SMALLFLOAT data type
built-in casts 2-64
description 2-43
display format 3-36, 3-38
SMALLINT data type
built-in casts 2-64
description 2-43
length (syscolumns) 1-32
Smart large objects
description 2-50
syscolattribs data 1-26
See also sbspacee.
Smart-large-object handles 3-87
Software dependencies 4
SOFTWARE registry key 3-13
SOME operator 2-70
Sorting
DBSPACETEMP environment variable 3-46
PSORT_DBTEMP environment variable 3-89
PSORT_NPROCS environment variable 3-89
See also Collation.
Sort-merge join 3-81
Space
DATETIME delimiter 2-18
INTERVAL delimiter 2-28
SPL routine 1-58, 2-60, 3-31, 3-34
SPL variables 2-60
SQL
new features 6
SQL character set 3-54
SQL Communication Area (SQLCA) 3-27
SQL (Structured Query Language) 3-27
sqlhosts file 3-61, 3-69, 3-70
SQLHOSTS subkey 3-71
SQLSTATE value 1-39
sqlwarn array 3-27
Stack size 1-60, 3-71
STACKSIZE configuration parameter 3-71
Staging area blobspace 3-68
Standard Graphic Markup Language (SGML) 2-14
START DATABASE statement 3-42
STAT data type 1-38
state table in stores_demo database A-8
state table in superstores_demo database B-19
Statement cache 3-92
Statements of SQL
ALTER OPTICAL CLUSTER 1-55
ALTER TABLE 1-12, 1-71
CLOSE 3-83
CONNECT 3-42, 3-65, 3-69
CREATE ACCESS METHOD 1-18
CREATE AGGREGATE 1-17
CREATE CAST 1-23, 2-66
CREATE DATABASE 3-42
CREATE DISTINCT TYPE 1-79, 2-24, B-22
CREATE EXTERNAL TABLE 1-40, 1-41
CREATE FUNCTION 3-80
CREATE IMPLICIT CAST B-22
CREATE INDEX 1-7, 1-46, 1-48, 1-51, 1-64, 1-71, 3-53
CREATE OPAQUE TYPE 1-79, 2-35
CREATE OPERATOR CLASS 1-53
CREATE OPTICAL CLUSTER 1-54, 1-55
CREATE PROCEDURE 1-57
CREATE ROLE 1-65, 1-76
CREATE ROUTINE FROM 1-65
CREATE ROW TYPE 1-79, 2-36
CREATE SCHEMA AUTHORIZATION 1-6
CREATE SYNONYM 1-66
CREATE TABLE 1-36, 1-63, 1-67
CREATE TRIGGER 1-75
CREATE VIEW 1-76
DATABASE 3-43
DECLARE 3-83
DELETE 1-13, 1-62, 1-77
DESCRIBE 3-61
DROP CAST B-22
DROP DATABASE 3-43
DROP FUNCTION 1-61
DROP INDEX 1-71
DROP OPTICAL CLUSTER 1-55
DROP PROCEDURE 1-61
DROP ROUTINE 1-61
DROP ROW TYPE 2-36
DROP TYPE 2-24, 2-35
DROP VIEW 1-81
FETCH 3-83
GET DIAGNOSTICS 1-39
GRANT 1-42, 1-65, 1-69, 1-81
INSERT 1-77, 2-60, 3-26, 3-34

LOAD 2-11, 2-44, 3-27, 3-35
OPEN 3-83
PREPARE 1-71
REVOKE 1-68, 1-76
SELECT 1-12, 1-37, 1-62, 3-47
SET OPTIMIZATION 3-84
SET PDQPRIORITY 3-86
SET SESSION AUTHORIZATION 1-61
SET STMT_CACHE 3-92
START DATABASE 3-43
UNLOAD 3-28, 3-35
UPDATE 2-11, 2-44, 3-26
UPDATE STATISTICS 1-13, 3-52
UPDATE STATISTICS FOR PROCEDURE 1-62
static option of ESQL/C 3-59
STMT_CACHE configuration parameter 3-92
STMT_CACHE environment variable 3-92
STMT_CACHE keyword 3-92
stock table in stores_demo database A-4
stock table in superstores_demo database B-20
stock_discount table in superstores_demo database B-21
Storage identifiers 3-54
Stored procedure language (SPL) 1-58, 2-60, 3-31
stores_demo database 5
call_type table columns A-7
catalog table columns A-5
customer table columns A-2
cust_calls table columns A-6
data values A-17
description A-1
items table columns A-4
join columns A-9
manufact table columns A-7
primary-foreign key relationships A-10 to ??
stock table columns A-4
structure of tables A-2
String Editor dialog box 3-14
strings option of gcc 3-63
Structured Query Language (SQL) 3-27
See also Statements of SQL.
Subscripts 2-12, 2-45
SUBSTRING function 1-12
Subtable 1-50, B-14, B-25
Subtype 1-50, 2-36
Summary
of data types 2-6
of environment variables, by topic 3-95
of environment variables, by type of server 3-18
of system catalog tables, by type of server 1-14
superstores_demo database 5
call_type table columns B-8
catalog table columns B-8
customer table columns B-10, B-11, B-12
cust_calls table columns B-9
description B-6
items table columns B-13
manufact table columns B-17
orders table columns B-14, B-15, B-16, B-18
primary-foreign key relationships B-25 to B-28
sales_rep table columns B-19
stock table columns B-20
stock_discount table columns B-21
structure of tables B-7
Supertable 1-50, B-14, B-25
Supertype 1-50, 2-36
Support function
routine identifier 1-58
Support functions 2-34, 2-59
Symbol table 1-58, 1-59
Synonym
syssynonyms data 1-66
syssyntable data 1-66
systables data 1-69
Syntax conventions
description of 10
sysaggregates system catalog table 1-17
sysams system catalog table 1-18
sysattrtypes system catalog table 1-21
sysblobs system catalog table 1-22
sysbuiltintypes table 1-6
syscasts system catalog table 1-23, 2-62
syschecks system catalog table 1-24
syscheckudrdep system catalog table 1-25
syscolattribs system catalog table 1-26
syscolauth system catalog table 1-27
syscoldepend system catalog table 1-28
syscolumns system catalog table 1-29
sysconstraints system catalog table 1-34
syscrd database 1-6
sysdefaults system catalog table 1-35
sysdepend system catalog table 1-37
sysdistrib system catalog table 1-37
sysdomains system catalog table 1-39
syserrors system catalog table 1-39
sysextcols system catalog table 1-40
sysextdfiles system catalog table 1-41
sysexternal system catalog table 1-41
sysfragauth system catalog table 1-42
sysfragments system catalog table 1-43
sysindexes system catalog table 1-46
sysindices system catalog table 1-48
sysinherits system catalog table 1-50
syslangauth system catalog table 1-50
syslogmap system catalog table 1-51
sysmaster database 1-6
initialization 3-69

versus system catalog tables 1-6
sysnewdepend system catalog table 1-51
sysobjstate system catalog table 1-52
sysopclasses system catalog table 1-53
sysopclstr system catalog table 1-54
sysprocauth system catalog table 1-56
sysprocbody system catalog table 1-57
sysprocedures system catalog table 1-58
sysprocplan system catalog table 1-62
sysreferences system catalog table 1-63
sysrepository system catalog table 1-64
sysroleauth system catalog table 1-65
sysroutinelangs system catalog table 1-65
syssynonyms system catalog table 1-66
syssyntable system catalog table 1-66
systabamdata system catalog table 1-67
systabauth system catalog table 1-68
systables system catalog table 1-69
System administrator (DBA) 1-6
System applet 3-15
System catalog
sysaggregates 1-17
sysams 1-18
sysattrtypes 1-21
sysblobs 1-22
syscasts 1-23
syschecks 1-24
syscheckudrdep 1-25
syscolattribs 1-26
syscolauth 1-27
syscoldepend 1-28
syscolumns 1-29
sysconstraints 1-34
sysdefaults 1-35
sysdepend 1-37
sysdistrib 1-37
sysdomains 1-39
syserrors 1-39
sysextcols 1-40
sysextdfiles 1-41
sysexternal 1-41
sysfragauth 1-42
sysfragments 1-43
sysindexes 1-46
sysindices 1-48
sysinherits 1-50
syslangauth 1-50
syslogmap 1-51
sysnewdepend 1-51
sysobjstate 1-52
sysopclasses 1-53
sysopclstr 1-54
sysprocauth 1-56
sysprocbody 1-57
sysprocedures 1-58
sysprocplan 1-62
sysreferences 1-63
sysrepository 1-64
sysroleauth 1-65
sysroutinelangs 1-65
syssynonyms 1-66
syssyntable 1-66
systabamdata 1-67
systabauth 1-68
systables 1-69
systraceclasses 1-72
systracemsgs 1-73
systrigbody 1-74
systriggers 1-75
sysusers 1-76
sysviews 1-76
sysviolations 1-77
sysxtddesc 1-78
sysxtdtypeauth 1-78
sysxtdtypes 1-79
System catalog tables
access methods 1-18, 1-67
accessing 1-12
altering contents 1-12
authorization identifiers 1-76
casts 1-23
columns 1-29
complex data types 1-21, 1-79
constraint violations 1-77
constraints 1-24, 1-28, 1-34
data distributions 1-37
database tables 1-69
default values 1-35
dependencies 1-37, 1-51
description 1-5
example
syscolauth 1-10
syscolumns 1-8
sysindexes 1-11
systabauth 1-10
systables 1-7
external tables 1-40, 1-41, 1-42
fragmentation 1-42, 1-43
indexes 1-46, 1-48, 1-64
inheritance 1-50
list of tables 1-14
messages 1-39, 1-73
operator classes 1-53
optical clusters 1-54
privileges 1-27, 1-42, 1-68, 1-76, 1-78
programming languages 1-50, 1-65
referential constraints 1-34, 1-63, 1-77
roles 1-65
routines 1-56, 1-57, 1-58, 1-62
sequences 1-69
simple large objects 1-22
smart large objects 1-26
synonyms 1-66, 1-67
trace classes 1-72
trace messages 1-73
triggers 1-74, 1-75
updating 1-13
use by database server 1-7
user-defined aggregates 1-17
user-defined data types 1-78, 1-79
views 1-70, 1-76
See also System catalog.
System control panel 3-18
System environment variable 3-18
System requirements
database 4
software 4
System security officer (DBSSO) 3-73

SYSTEM() command, on NT 3-74
systraceclasses system catalog
table 1-72
systracemsgs system catalog
table 1-73
systrigbody system catalog
table 1-74
systriggers system catalog
table 1-75
sysusers system catalog table 1-76
sysutils database 1-6
sysuuid database 1-6
sysviews system catalog table 1-76
sysviolations systems catalog
table 1-77
sysxtddesc system catalog
table 1-78
sysxtdtypeauth system catalog
table 1-78
sysxtdtypes system catalog
table 1-79, 2-35

T

tabid, description 1-8, 1-71
Table
changing a column data type 2-62
dependencies, in sysdepend 1-37
fragmented 1-43
hashing parameters 1-67
hierarchy 1-50, 2-37, B-25
inheritance, sysinherits data 1-50
lock mode 3-56
nonfragmented 3-53
separate from large object
storage 2-48
structure in superstores_demo
database B-7
synonyms in syssyntable 1-66
system catalog tables 1-17 to 1-79
temporary 3-46, 3-48
temporary in SE 3-48
typed, and named ROW
type 2-37
untyped, and unnamed
ROW 2-38
Table-based fragmentation 1-44
Table-level privilege
PUBLIC 1-81
sysfragauth data 1-42
systabauth data 1-10, 1-68
Tape management
setting DBREMOTECMD 3-45
Temporary dbspace 3-46
Temporary files 3-48
in SE, specifying directory with
DBTEMP 3-48
setting DBSPACETEMP 3-46
setting PSORT_DBTEMP 3-89
Temporary tables 3-46
in SE, specifying directory with
DBTEMP 3-48
specifying dbspace with
DBSPACETEMP 3-46
TERM environment variable 3-92
TERMCAP environment
variable 3-93
termcap file
setting INFORMIXTERM 3-72
setting TERMCAP 3-93
Terminal handling
setting INFORMIXTERM 3-72
setting TERM 3-92
setting TERMCAP 3-93
setting TERMINFO 3-94
terminfo directory 3-72, 3-94
TERMINFO environment
variable 3-94
TEXT data type
casting to CLOB 2-45
collation 2-45
description 2-44
increasing buffer size 3-28
inserting values 2-44
length (syscolumns) 1-33
nonprintable characters 2-45
queries 2-45
restrictions
in Boolean expression 2-44
with GROUP BY 2-44
with LIKE or MATCHES 2-44
with ORDER BY 2-44
setting buffer size 3-28
sysblobs data 1-22
with control characters 2-44
Text editor 3-35
Thousands separator 2-32
thread flag of ESQL/C 3-94
THREADLIB environment
variable 3-94
Time data types
arithmetic 2-51
length (syscolumns) 1-33
listed 2-48
time table in sales_demo
database B-6
Time values
DBCENTURY setting 3-28
DBDATE setting 3-32
DBTIME setting 3-48
GL_DATETIME settings 3-52
USEOSTIME parameter 2-20
Tip icons 8
TO keyword
DATETIME qualifier 2-17
EXTEND function 2-54
INTERVAL qualifier 2-26
TODAY operator 1-36
Trace class 1-72
Trace statements 1-73
Transaction isolation level 1-86,
3-81
Transaction logging 1-26, B-2
Trigger
creation-time value 3-31, 3-34
sysobjstate data 1-52
systrigbody data 1-74
systriggers data 1-75
TRUE setting
BOOLEAN values 2-10
CPFIRST 3-25
ISM_COMPRESSION 3-75
ISM_ENCRYPTION 3-76
sysams table 1-19
Truncation 2-12
TYPE keyword 2-37

U

UDA *See* User-defined aggregates.
UDR *See* User-defined routine.
UDT *See* User-defined data type.
Unnamed ROW data type
See ROW type.

Unary arithmetic operators 2-70
Uncommitted read 1-86
Under privilege 1-68
Unique constraint 2-39, 2-40
Unique index 1-46, 2-39, 2-40
Unique keys 1-19
Unique numeric values
  SERIAL data type 2-39
  SERIAL8 data type 2-40
  See also Sequential integers.
UNITS operator 2-16, 2-52, 2-56, 2-70
units table in superstores_demo database B-22
UNIX
  BSD, default print utility 3-45
  environment variables 3-5
  PATH environment variable 3-84
  System V
    default print utility 3-45
    terminfo libraries 3-72, 3-94
  temporary files 3-89
  TERM environment variable 3-92
  TERMCAP environment variable 3-93
  TERMINFO environment variable 3-94
UNIX operating system
  default locale for 5
UNLOAD statement 3-28, 3-35
Unnamed ROW data type
  declaring 2-38
  description 2-37
  inserting values 2-38
unset utility 3-10
unsetenv utility 3-10
Unsetting an environment variable 3-10
Untyped table 1-70
Update privilege 1-27, 1-43, 1-68, 3-80
UPDATE statement 1-77, 2-11, 2-44, 3-61
UPDATE STATISTICS FOR PROCEDURE statement 1-62
UPDATE STATISTICS statement 3-52
  and DBUPSPACE environment variable 3-52
  effect on sysdistrib table 1-38
  sysindices data 1-49
  sysindices (index statistics) 1-55
  update system catalog 1-13
Update trigger 1-75
Uppercase mode codes 1-61
Uppercase privilege codes 1-10, 1-27, 1-42, 1-68, 1-78
Usage privilege 1-78
USEOSTIME configuration parameter 2-20
User environment variable 3-18
User informix 1-12, 1-23
User name 1-86
User privileges
  syscolauth data 1-27
  sysfragauth data 1-42
  syslangauth data 1-50
  sysprocauth data 1-56
  systabauth data 1-68
  sysusers data 1-76
  sysxtdtypeauth data 1-78
User-defined aggregates 1-17
User-defined casts 1-23, 2-66
User-defined data type
  casting 2-66
  casting into built-in type 2-62
  opaque 2-61
  sysxtddesc data 1-78
  sysxtdtypes data 1-78, 1-79
  UDT indexes 3-53
User-defined routine
  casts (syscasts) 1-23
  check constraints (syscheckudrdep) 1-25
  error messages (syserrors) 1-39
  for opaque data types 2-35
  language authorization (syslangauth) 1-50
  privileges 1-56, 3-80
  protected 1-61
  secondary access method 1-34
  sysprocedures data 1-58
Utility
  archecker 3-24
  chkenv 3-8, 3-11
  DB-Access 1-12, 1-81, 3-7, 3-27, 3-36, 3-68, B-2
  dbexport 3-35
  dbload 2-10, 2-11, 2-44
  dbschema 1-40, 1-41, 1-42, 1-61
  dce_login 3-67
  env 3-11
  export 3-9
  gcc 3-63
  getenv 3-6
  ifx_getenv 3-7
  ifx_putenv 3-7
  imcadmin 3-61
  load 1-42
  lp 3-44
  lpr 3-44
  MaxConnect 3-62
  ON-Bar 3-74, 3-75, 3-76
  oninit 3-57
  onload 2-10, 2-11, 2-44
  onpload 3-41, 3-88
  onxfer 3-95
  printenv 3-11
  putenv 3-6
  regedt32.exe 3-13
  set 3-16
  setenv 3-10
  Setnet32 3-7
  source 3-9
  unset 3-10
  unsetenv 3-10, 3-54
  vi 3-36

V

VARCHAR data type
  collation 2-47
  conversion to NVARCHAR 3-40, 3-41
  description 2-46
  length (syscolumns) 1-32
  multibyte characters 2-47
  nonprintable characters 2-47
  storing numeric values 2-47
  See also CHARACTER VARYING data type.
Variable-length UDT 1-79
VARIANT routine 1-60
vi text editor 3-36
View
  columns view 1-83

Information Schema 1-80
server_info view 1-85
sql_languages view 1-84
sysdepend data 1-37
sysindexes view 1-49
syssynonyms data 1-66
syssyntable data 1-66
systabauth data 1-68
systables data 1-69
sysviews data 1-76
tables view 1-82
Violations
sysobjstate data 1-52
sysviolations data 1-77
Virtual machine 3-24, 3-78
Virtual processor 3-91

W

Warning icons 8
Warning message 1-40, 3-27
WHERE keyword 1-12, 1-24, 2-45
Whitespace in identifiers 3-53
Window borders 3-72
Windows environments
manipulating environment variables 3-13
setting environment variables 3-7
Windows NT
default locale for 5
Windows registry 3-13

X

X setting
sysams.am_sptype 1-18
systabauth.tabauth 1-68
XBSA
debugging records 3-75
message log file 3-75
shared library 3-75
XFER_CONFIG environment variable 3-95
xfer_config file 3-95
XML (Extensible Markup Language) 2-14
XOR setting 3-76
XPG4 standard 1-83
XPS (Extended Parallel Server) 1-14, 3-18, B-1
X/Open
compliance 1-85
Information Schema views 1-80
server_info view 1-85
X/Open compliance level 15

Y

Y setting
DBDATE 3-32
DBTIME 3-50
sysroleauth.is_grantable 1-65
Year 2000 3-29
YEAR keyword
DATETIME qualifier 2-17
EXTEND function 2-54
INTERVAL qualifier 2-27
Year values, two and four digit 2-19, 3-28, 3-32, 3-50
YES setting
columns.is_nullable 1-83
sql_languages.integrity 1-84
yes setting
NODEFDAC 3-80

Z

Zero (0)
C null as terminator 2-47
DBDATE separator 3-33
DECIMAL scale 2-21
hexadecimal digit 3-35
IFX_DIRECTIVES setting 3-58
IFX_LONGID setting 3-59
IFX_NETBUF_PVTPOOL_SIZE setting 3-60
INFORMIXOPCACHE setting 3-68
integer scale 1-84, 2-21
ISM_DEBUG_LEVEL setting 3-75
OPTCOMPIND setting 3-81
OPTMSG setting 3-82
OPTOFC setting 3-83
padding of 1-digit years 3-29
padding with DBFLTMASK 3-36
padding with DBTIME 3-51
PDQPRIORITY setting 3-85
PSORT_NPROCS setting 3-91
STMT_CACHE setting 3-92
sysams values 1-19
sysfragments.hybdpos 1-45
sysindices.nrows 1-48
systables.type_xid 1-70
sysxdtypes values 1-79
zip column B-16
zipcode column A-2, B-15

Symbols

(-), hyphen
DATE separator 3-32
DATETIME delimiter 2-18
INTERVAL delimiter 2-28
subtraction operator 2-52, 2-70
symbol in syscolauth 1-10, 1-27
symbol in sysfragauth 1-42
symbol in systabauth 1-69
unary operator 2-53, 2-70
(!=), not equal to
relational operator 2-70
($), dollar sign
currency symbol 2-32, 3-38
pathname indicator 3-24, 3-84
(%), percentage
DBTIME escape symbol 3-49
pathname indicator 3-24, 3-47
('), single quotes
string delimiter 3-39, 3-54
(( )), parentheses
delimiters in expressions 2-56
( ), blank space
DATETIME delimiter 2-18
INTERVAL delimiter 2-28
padding CHAR values 2-13
padding VARCHAR values 2-47
(*), asterisk
multiplication operator 2-41, 2-52, 2-57, 2-70
systabauth value 1-10, 1-69
wildcard symbol 1-24, 1-81
(+), plus sign
addition operator 2-52, 2-70
truncation indicator 3-59

unary operator 2-70
(,), comma
decimal point 3-39
list separator 2-33, 2-37, 3-46
thousands separator 2-32
(.), period
DATE separator 3-33
DATETIME delimiter 2-18
decimal point 2-22, 2-32, 3-39
execution symbol 3-9
INTERVAL delimiter 2-28
membership operator 2-70
nested dot notation 2-60
(/), slash
DATE separator 2-16, 2-56, 3-33
division operator 2-52, 2-70
pathname delimiter 3-12, 3-42, 3-79
(:), colon
cast (::) operator 2-66, 2-70
DATETIME delimiter 2-18
INTERVAL delimiter 2-28
list separator 3-46, 3-54, 3-66, 3-79, 3-84
(;), semicolon
list separator 3-66, 3-84
(<), less than
angle (< >) brackets 2-12, A-30
relational operator 1-12, 2-70, 3-34
(=), equality
assignment operator 3-16
relational operator 1-24, 2-10, 2-15, 2-70
(>), greater than
angle (< >) brackets 2-12, A-30
relational operator 1-12, 2-70
([ ]), brackets
MATCHES range delimiters 2-48
substring operator 2-12, 2-45, 2-70
(\ ), backslash
invalid as delimiter 3-35
pathname delimiter 3-16, 3-71
(\#), sharp
comment indicator 3-8
(_), underscore
in SQL identifiers 3-53
({ }), braces
collection delimiters 2-30, 2-33, 2-42, 2-43, 2-60
pathname delimiters 3-11
(|), vertical bar
absolute value delimiter 2-25
concatenation (| |) operator 2-70
field delimiter 3-35
(~), tilde
pathname indicator 3-12
(”), double quotes
delimited SQL identifiers 3-54
string delimiter 2-30, 2-33, 2-43